前　言

MATLAB 作为当今工程界最流行的计算软件，其应用范围涵盖了控制、电子、通信、航空、航天、航海等众多领域。在项目开发和系统设计过程中，使用 MATLAB 可以提高设计速度、缩短设计时间、减少试验次数、降低研究成本，所以，熟练掌握 MATLAB 已经成为从事工程设计研究必须具备的基本技能。因此，目前国内绝大多数重点大学都已经开设了 MATLAB 课程。

现在有关 MATLAB 的书籍种类很多，但是由于侧重点和面向的读者不同，存在一定的局限，有的只涉及最基本的知识，不能为工程实践提供较多帮助；有些则针对专业人士，内容艰深，应用面很窄；有的只是英文原版的翻译，罗列的命令很多，举的例子较少，不便于读者自学，作为手册，还可胜任，作为教材，有些勉强。在作者的教学实践中，深深感到寻找一本合适的 MATLAB 教材有一定的困难，这也促使作者下决心为在校学生量身定做一本合适的 MATLAB 教材。

本书在撰写的过程中，听取了很多学习 MATLAB 课程的学生的意见，综合了身处教学一线老师的体会和自己的教学经验，切实考虑了在校生的知识结构和水平，精心选择内容，合理安排布局，兼顾了一定的深度和广度。在书中，作者结合自己的教学、科研工作，编写了大量的实例，基本覆盖了从事控制工程所需的 MATLAB 知识，使学生在校内就可以接触到一定的工程实际内容，从而能更好地从事实际科研工作，为今后参加工作打好扎实的基础。同时教材的各章中均配备课后上机练习题，方便学生实践并检验自己的学习效果。

与其他 MATLAB 书籍相比，本书具有以下特点：① 内容新颖，主要讲述 MATLAB 7.x 版本；② 内容剪裁得当，兼顾教学与科研，基本覆盖了从事工程研究所需的 MATLAB 知识；③ 讲述深入浅出，实例选择得当，注重与专业知识的结合。全书的内容安排由浅入深，有助于读者了解更专业的知识。

本书第一章对 MATLAB 做了简单的介绍;第二章介绍 MATLAB 的基本操作,包括数值及变量、赋值语句、常用数学函数及 MATLAB 语言中的关系与逻辑运算;第三章内容为数组与矩阵的运算;第四章讲解多项式的表达式及其操作;第五章介绍 MATLAB 中的字符串和其他数据类型;第六章阐述 MATLAB 的基本绘图功能;第七章介绍 M 文件与 M 函数;第八章介绍基于 Simulink 的动态系统仿真方法;第九章讲述 Simulink 高级仿真技术;第十章讨论如何用 S-函数扩展 Simulink 功能;第十一章介绍 Simulink 命令行仿真技术;第十二章介绍如何利用状态流 Stateflow 进行控制系统状态转换;第十三章讲解如何应用控制系统工具箱分析设计系统。

本书的写作分工是第一、二、四章由栾云凤执笔,第五章至第十二章由贾秋玲执笔,第三、十三章由袁冬莉执笔。

在从创意、酝酿到完成全稿的写作过程中,我们得到了西北工业大学自动化学院信息与控制专业各位老师的大力支持和鼓励,同时也得到了很多有效的建议,在此深表谢意。

由于编者水平有限,书中错误在所难免,欢迎读者批评指正。

编　者

2005 年 9 月

目　录

第一章　MATLAB 简介 ………………………… 1

1.1　MATLAB 简介 ………………………… 1
1.2　MATLAB 的安装 ………………………… 2
1.3　MATLAB 的工作环境 ………………………… 3
1.4　MATLAB 的系统命令 ………………………… 4
1.5　在线帮助 ………………………… 5
习题 ………………………… 7

第二章　MATLAB 的基本操作 ………………………… 8

2.1　数值及变量 ………………………… 8
2.2　MATLAB 赋值语句 ………………………… 9
2.3　MATLAB 常用数学函数 ………………………… 10
2.4　MATLAB 语言中的关系与逻辑运算 ………………………… 11
习题 ………………………… 13

第三章　数组与矩阵的运算 ………………………… 15

3.1　数组和矩阵的产生 ………………………… 15
3.2　矩阵的子矩阵的寻访与赋值 ………………………… 17
3.3　矩阵的运算 ………………………… 19
3.4　矩阵函数 ………………………… 20
习题 ………………………… 24

第四章　多项式的表达式及其操作 ………………………… 25

4.1　多项式的表达和生成 ………………………… 25
4.2　多项式运算函数 ………………………… 26
习题 ………………………… 27

第五章 字符串和其他数据类型 …… 28

5.1 字符串及其处理 …… 28
5.2 元胞数组 …… 31
5.3 结构体数组 …… 34
习题 …… 36

第六章 MATLAB 基本绘图功能 …… 37

6.1 二维图形的绘制 …… 37
6.2 三维图形的绘制 …… 45
习题 …… 47

第七章 M 文件与 M 函数 …… 49

7.1 M 文件编辑器 …… 49
7.2 MATLAB 语言的语法 …… 50
7.3 脚本文件和 M 函数的编写与调用 …… 58
习题 …… 60

第八章 基于 Simulink 的动态系统仿真入门 …… 61

8.1 启用 Simulink 并建立系统模型 …… 61
8.2 Simulink 模型库简介 …… 61
8.3 Simulink 模型的构建 …… 73
8.4 基于 Simulink 系统仿真技术应用举例 …… 80
8.5 Simulink 的子系统技术 …… 87
8.6 Simulink 的调试技术 …… 101
习题 …… 108

第九章 Simulink 高级仿真技术 …… 111

9.1 Scope 模块的高级使用技术 …… 111
9.2 Simulink 的工作原理 …… 115
9.3 系统过零的概念与解决方案 …… 118
9.4 系统代数环的概念与解决方案 …… 122
9.5 高级积分器 …… 125
习题 …… 129

第十章 用 S-函数扩展 Simulink …… 130

10.1 S-函数概述 …… 130
10.2 S-函数工作原理 …… 134
10.3 编写 M 文件 S-函数 …… 136

10.4 编写 C MEX S-函数 …… 144
习题 …… 154

第十一章 Simulink 命令行仿真技术及回调函数概念 …… 156

11.1 Simulink 与 MATLAB 的接口 …… 156
11.2 使用命令行方式对动态系统进行建模和仿真分析 …… 162
11.3 使用 MATLAB 脚本文件分析系统 …… 170
11.4 回调函数 …… 172
习题 …… 175

第十二章 利用状态流 Stateflow 进行控制系统状态转换 …… 176

12.1 有限状态机简介 …… 176
12.2 Stateflow 应用基础 …… 177
12.3 利用 Stateflow 完成复杂的状态逻辑判断及其动作 …… 214
习题 …… 225

第十三章 反馈控制系统的数学模型及设计工具 …… 226

13.1 数学模型的表示方法 …… 226
13.2 模型的基本结构 …… 230
13.3 不同模型对象的相互转换和模型数据的还原 …… 232
13.4 控制系统分析与设计 …… 233
习题 …… 243

参考文献 …… 245

第一章 MATLAB简介

本章主要介绍MATLAB计算软件的基本用途、功能、安装方法、启动界面以及一些系统命令，为后面章节的学习做准备。

1.1 MATLAB简介

MATLAB计算软件（以下简称MATLAB）产品家族是美国MathWorks公司开发的用于概念设计、算法开发、建模仿真、实时实现的理想的集成环境。自1980年问世以来，由于其完整的专业体系和先进的设计开发思路，使得MATLAB在众多领域都有着广阔的应用空间，特别是在MATLAB的主要应用方面——科学计算、建模仿真以及信息工程系统的设计开发上已经成为行业内的首选设计工具，广泛应用于生物医学工程、图像信号处理、语言信号处理、信号分析、电信、时间序列分析、控制论和系统论等各个领域。

由于使用MATLAB编程运算与人类进行科学计算的思路和表达方式完全一致，所以学习MATLAB不像学习其他高级语言，如Basic,Fortran和C等那样难于掌握。MATLAB具有用法简易、运用灵活、程序结构性强且兼具延展性等特点。

MATLAB的含义是矩阵实验室（Matrix Laboratory），其名字来自“matrix”和“laboratory”两个词的前三个字母的组合。主要用于矩阵的方便存取，其基本元素是无须定义维数的矩阵。MATLAB自问世以来，就是以数值计算见长。MATLAB具有很强的数值运算功能，在MATLAB环境中，有超过500种数学、统计、科学及工程方面的函数可使用。MATLAB进行数值计算的基本单位是复数数组（或称阵列），这使得MATLAB高度“向量化”。经过十多年的完善和扩充，MATLAB现已发展成为线性代数课程的标准工具。由于它不须定义数组的维数，并给出了矩阵函数、特殊矩阵专门的库函数，使之在求解诸如信号处理、建模、系统识别、控制、优化等领域的问题时，显得大为简捷、高效、方便，这是其他高级语言所无法比拟的。

在MATLAB产品体系的演化历程中最重要的一个体系变更是引入了Simulink。Simulink是在MATLAB仿真平台下的一种图形仿真工具，它可以和MATLAB通过求解器进行无缝连接。Simulink是一个进行动态系统建模、仿真和综合分析的集成软件包。其框图化的设计方式和良好的交互性，对工程人员本身的计算机操作与编程的熟练程度的要求降到了最低，工程人员可以把更多的精力放到理论和技术的创新上去。

针对控制逻辑的开发、协议栈的仿真等要求，MathWorks公司在Simulink平台上还提供

了用于描述复杂事件驱动系统的逻辑行为的建模仿真工具——Stateflow。通过 Stateflow，用户可以用图形化的方式描述事件驱动系统的逻辑行为，并无缝地结合到 Simulink 的动态系统仿真中。

在 MATLAB/Simulink 基本环境之上，MathWorks 公司为用户提供了丰富的扩展资源，这就是大量的 Toolbox 和 Blockset。在从 1985 年推出第一个版本以后的近 20 年发展过程中，MATLAB 已经从单纯的 Fortran 数学函数库演变为多学科、多领域的函数包或模块库的提供者。这些 Toolbox 和 Blockset 是为了解答特殊类型的问题而扩展的 MATLAB 环境下的 MATLAB 函数(M 文件)的综合，给用户提供了特别应用领域所需的许多函数，可用来求解各类学科的问题。现有工具箱有符号运算(利用 Maple V 的计算核心执行)、图像处理、统计分析、信号处理、神经网络、控制系统辨识、模拟分析、最优化、模糊逻辑、μ 分析及综合、化学计量分析等。随着 MATLAB 版本的不断升级，其所含的工具箱的功能也越来越丰富，因此，MATLAB 的应用范围也越来越广泛，成为涉及数值分析的各类工程师不可缺少的工具。

MATLAB 中包括了图形界面编辑 GUI，改变了以前单一的在指令窗通过文本形式的指令进行各种操作的状况，使用者可以像使用 VB，VC，VJ 和 DELPHI 软件那样进行一般的可视化的程序编辑。

为了将系统开发流程和工程实现进行有效的连接，也为了使系统级的设计产物和硬件产品直接挂钩。MATLAB 产品体系中加入了连接工程实现的桥梁——实时代码生成工具 Real - Time Workshop(RTW)，称之为实时工作间。RTW 使用户可以直接将 Simulink 框图模型转化为实时标准 C 代码，进而为快速原型系统、半物理仿真系统或产品提供设计输入。

本教材主要陈述 MATLAB 7. x 的基本功能，教会读者如何运用 MATLAB 7. x 进行科学计算，如何利用 MATLAB 7. x 进行简单及复杂动态系统的分析、仿真和设计等。

MATLAB 7. x 是用于分析数据、开发算法并进行应用的一种高级技术语言和开发环境。相比早先的 MATLAB 版本，MATLAB 7. x 针对编程环境、代码效率、数据可视化、数学计算、文件 I/O 等方面进行了升级。

1.2 MATLAB 的安装

MATLAB 只有在适当的环境下才能正常运行。因此适当配置外部系统是保证 MATLAB 运行的先决条件。MATLAB 本身适用于许多机种和系统。在 PC 机上安装 MATLAB 的步骤是非常简单的，一般来说，在光盘插入光驱后，会自动启动“安装向导”。如果没有自动启动，可以运行该光盘上的 setup. exe 应用程序来启动“安装向导”。在安装过程中出现的所有界面都是标准的，用户只要按照屏幕提示操作即可。

1.3 MATLAB的工作环境

运行桌面上的MATLAB图标或它的可执行文件，就会自动创建MATLAB默认的窗口，如图1.1所示。对MATLAB窗口说明如下：

1. **命令窗口**

MATLAB命令窗口(Command Window)用来输入变量或运行函数MATLAB(.m)文件。

例如，若输入

```
>>A = [1 2 3; 4 5 6; 7 8 10]
```

按下回车键后命令窗口显示如下：

```
A =
    1 2 3
    4 5 6
    7 8 10
```

用户可用clc清除命令窗口，只清除了显示，并不清除工作空间，仍可按向上的箭头看到以前发出的命令。

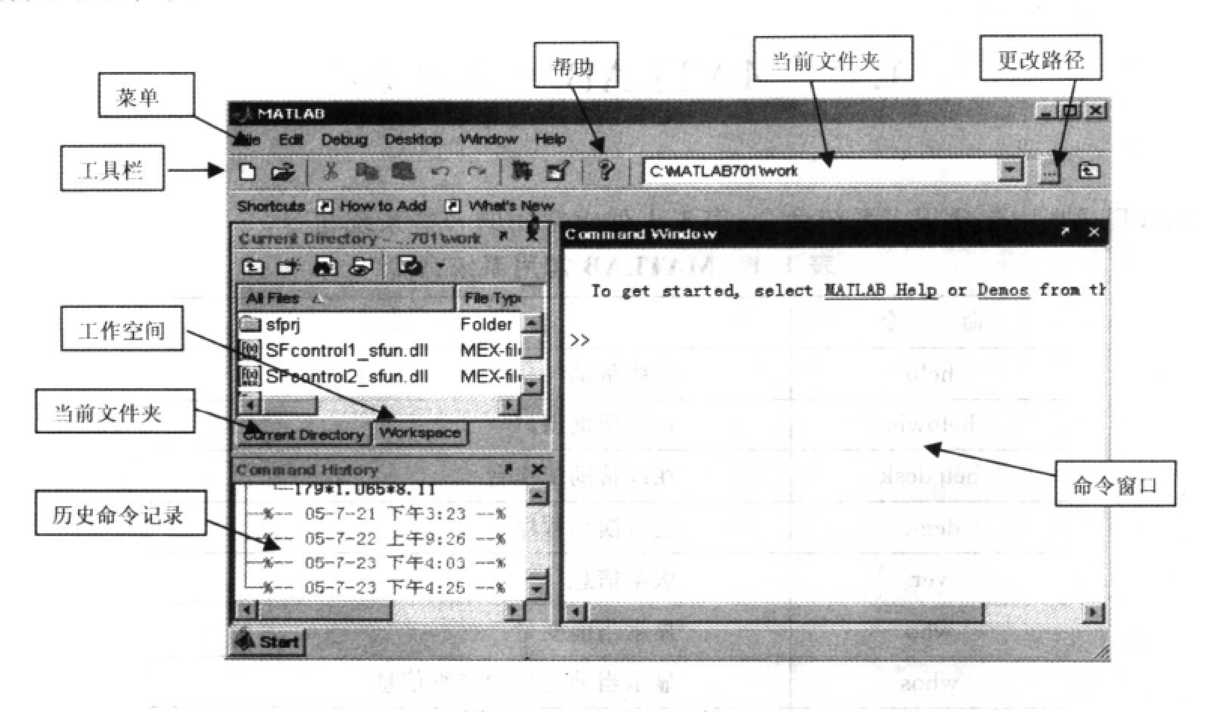


图1.1 MATLAB窗口

2. **六个功能菜单**

这六个功能菜单(File，Edit，Debug，Desktop，Window，Help)各自有下一层的功能。File和Edit实现对文件的管理和编辑；Help可检索帮助信息；Debug是调试程序工具；Desktop及Window可设置MATLAB窗口显示界面的形式及窗口；MATLAB的菜单及选择方式与

Windows 下各种软件环境中的文件管理方式相同。

3. 工作空间

MATLAB 的工作空间(Workspace)主要用来存放变量。

用户可以在 MATLAB 命令窗口中键入 who 或 whos 来查看当前工作空间中有哪些变量,如

```
>>whos
```

按下回车键后命令窗口显示如下:

```
Name      Size      Bytes   Class
  A        3x3        72    double array
Grand total is 9 elements using 72 bytes
```

用户可以采用 clear 命令删除工作空间中的所有变量。

4. 图形窗口

在命令窗口键入 figure,可产生一个与命令窗口隔离的图形窗口。如在命令窗口键入如下命令:

```
t = 0:pi/100:2 * pi;
y = sin(t);
plot(t,y)
```

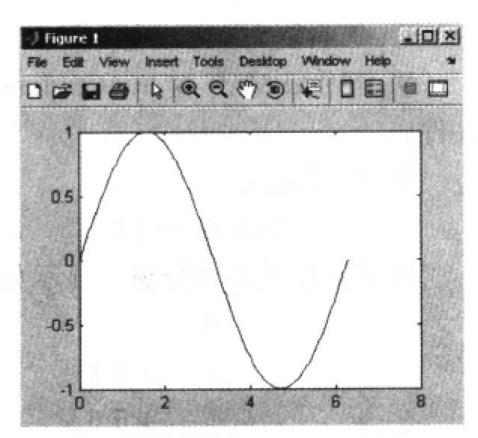


图 1.2 MATLAB 图形窗口

plot 函数则会在图形窗口中绘制图形,见图 1.2 所示。

1.4 MATLAB 的系统命令

MATLAB 中包含很多系统命令,表 1.1 列出常用的一些系统命令。

表 1.1 MATLAB 常用系统命令

命令	含义
help	在线帮助
helpwin	在线帮助窗口
helpdesk	在线帮助工作台
demo	运行演示程序
ver	版本信息
who	显示当前变量
whos	显示当前变量的详细信息
clear	清空工作空间的变量和函数
pack	整理工作空间的内存
load	把文件中的变量调入到工作空间
save	把变量存入文件中

续表

命　　令	含　　义
quit/exit	退出 MATLAB
what	显示指定的 MATLAB 文件
lookfor	在 HELP 里搜索关键字
which	定位函数或文件
path	获取或设置搜索路径
echo	命令回显
cd	改变当前的工作目录
pwd	显示当前的工作目录
dir	显示目录内容
unix	执行 unix 命令
dos	执行 dos 命令
!	执行操作系统命令
computer	显示计算机类型
figure	打开图形窗口
clc	清除命令窗口中的内容
close all	关闭所有图形窗口

1.5　在线帮助

在表 1.1 中所介绍的 MATLAB 系统命令中，help 和 lookfor 指令对于新老用户都是非常重要的，使用 help 和 lookfor 是获得在线帮助的最简单、有效的途径。help 可以列出用户所寻求帮助的函数的功能描述；而 lookfor 则列出所有函数的功能描述中含有用户所输入内容的函数简介。

例如，在 MATLAB 命令窗口键入 help inversion 和 lookfor inversion，命令窗口给出如下不同的提示：

```
>> help inversion
inversion.m not found.
>> lookfor inversion
COND    Condition number with respect to inversion.
SSINV State-space inversion.
utInv.m: % Metadata management for model inversion
```

AP Affine projection FIR adaptive filter using direct matrix inversion.

InversionFilter. m：%INVERSIONFILTER

TABLEINVERSION Table inversion routine for two d lookup tables.

InversionFilter. m：%INVERSIONFILTER

tableinversion. m：%TABLEINVERSION

如果用户想了解 MATLAB 中有什么指令可以调用，运行不带任何限定的 help，可以得到分类名称明细表。例

```
>> help
HELP topics:
MATLAB\general      — General purpose commands.
MATLAB\ops          — Operators and special characters.
MATLAB\lang         — Programming language constructs.
MATLAB\elmat        — Elementary matrices and matrix manipulation.
MATLAB\elfun        — Elementary math functions.
MATLAB\specfun      — Specialized math functions.
MATLAB\matfun       — Matrix functions — numerical linear algebra.
……
```

采用 help topic 指令形式可以获得具体子类的指令明细，其中 topic 代表子类的名称。如果用户想知道有关初等矩阵和矩阵操作指令一览表，可以运行以下指令。

```
>> help MATLAB\elmat
  Elementary matrices and matrix manipulation.
   Elementary matrices.
    zeros       — Zeros array.
    ones        — Ones array.
    eye         — Identity matrix.
  ……
  Basic array information.
    size        — Size of array.
    length      — Length of vector.
  ……
  Matrix manipulation.
    cat         — Concatenate arrays.
    reshape     — Change size.
    diag        — Diagonal matrices and diagonals of matrix.
  ……
  Multi—dimensional array functions.
    ndgrid      — Generate arrays for N—D functions and interpolation.
    permute     — Permute array dimensions.
  ……
```

```
Array utility functions.
   isscalar        — True for scalar.
   isvector        — True for vector.
……
Special variables and constants.
   ans             — Most recent answer.
   eps             — Floating point relative accuracy.
……
Specialized matrices.
   compan          — Companion matrix.
   gallery         — Higham test matrices.
……
```

在命令窗口中键入 computer 可以显示计算机类型，如

```
>>computer
ans =
    PCWIN
```

习　题

1.1　熟悉下列 MATLAB 常用系统命令的用途：help，ver，clc，clear，figure，close，who 等。

第二章 MATLAB 的基本操作

本章讲述 MATLAB 的基本操作，包括介绍 MATLAB 中数值及变量的表示、赋值语句、常用的数学函数和 MATLAB 语言中的关系与逻辑运算。

2.1 数值及变量

2.1.1 数值的记述

数值是科学计算中最重要的元素。MATLAB 的数值采用十进制表示，可以带小数点或负号。以下记述都是合法的：

3　　−99　　0.002　　1.3e−2　　4.5e33

MATLAB 的所有计算都以双精度格式计算。任何 MATLAB 的语句的执行结果都可以在屏幕上显示，同时赋值给指定的变量，没有指定变量时，赋值给一个特殊的变量 ans，数据的显示格式由 format 命令控制。

format 只是影响结果的显示，不影响其计算与存储；MATLAB 总是以双字长浮点数（双精度）来执行所有的运算。缺省的数据显示格式为短格式，只显示含 4 位小数的十进制数。

Format long　　长格式计数法，显示含 14 位小数的十进制数

Format long e　　长格式（科学）计数法

Format short e　　短格式（科学）计数法

例如，在 MATLAB 命令窗口中键入数据 12.3，有以下几种格式：

format (short)：　短格式（5 位定点数）12.3000

format long：　长格式（15 位定点数）12.30000000000000

format short e：　短格式 e 方式 1.2300e+001

format long e：　长格式 e 方式 1.230000000000000e+001

format bank：　2 位十进制 12.30

format hex：　十六进制格式

用户可以在命令窗口键入 help format，详细了解数据显示格式的种类。

2.1.2 变量

用户在 MATLAB 编程过程中需要定义很多变量。MATLAB 的变量命名必须遵循下列

3 个命名规则：

(1) 变量名中的英文字母大小写是有区别的，如 myvar 和 MyVar 表示两个不同的变量。

(2) 变量名的第一个字符必须为英文字母，不能超过 31 个字符，如 myvar31。

(3) 变量名可以包含下连字符、数字，但不能包含空格符、标点，如 my_var_31 是合法的。

MATLAB 中含有如表 2.1 所示的预定义变量。当 MATLAB 启动时，这些变量就会自动产生。

表 2.1　MATLAB 的预定义变量

预定义变量	说　　明
ans	预设的计算结果的变量名
eps	MATLAB 定义的正的极小值=2.2204e−16
pi	内建的 π 值
inf	∞值，无限大($\frac{1}{0}$)
NaN	非数，无法定义一个数目($\frac{0}{0}$)
i 或 j	虚数单位 $i=j=\sqrt{-1}$
nargin	函数输入参数个数
nargout	函数输出参数个数
realmax	最大的正实数
realmin	最小的正实数
flops	浮点运算次数

2.2　MATLAB 赋值语句

MATLAB 中书写表达式的规则与“手写算式”的方法几乎完全相同，具体规则如下：

(1) 表达式由变量名、运算符和函数名组成；

(2) 表达式将按常规的优先级从左向右运算。

如果一个指令过长可以在结尾加上“...”(表示此行指令与下一行连续)，例如 4 * sin(0.3) * 8 可以输入为

```
>>4 * sin(0.3) * ...
8
```

则输出为

```
ans =
    9.4566
```

2.3 MATLAB常用数学函数

在 MATLAB 的命令窗口中键入 help elfun，可以看到 MATLAB 常用的基本函数库。MATLAB 常用的基本函数库和其他常用函数如表 2.2～2.6 所示。

1. 三角函数和双曲函数

表 2.2 三角函数和双曲函数库

名称	含义	名称	含义	名称	含义
sin	正弦	csc	余割	artanh	反双曲正切
cos	余弦	arcsec	反正割	arcoth	反双曲余切
tan	正切	arccsc	反余割	sech	双曲正割
cot	余切	sinh	双曲正弦	csch	双曲余割
arcsin	反正弦	cosh	双曲余弦	arcsech	反双曲正割
arccos	反余弦	tanh	双曲正切	arcsch	反双曲余割
arctan	反正切	coth	双曲余切	arctan2	四象限反正切
arccot	反余切	arcsinh	反双曲正弦		
sec	正割	arccosh	反双曲余弦		

2. 指数和对数函数

表 2.3 指数和对数函数库

名称	含义	名称	含义	名称	含义
exp	以 e 为底的指数	log10	以 10 为底的对数	pow2	2 的幂
log	自然对数	log2	以 2 为底的对数	sqrt	平方根

3. 复数函数

表 2.4 复数函数库

名称	含义	名称	含义	名称	含义
abs	绝对值	conj	复数共轭	real	复数实部
angle	相角	imag	复数虚部		

4. 圆整函数和求余函数

表 2.5 圆整函数和求余函数库

名 称	含 义	名 称	含 义
ceil	向+∞圆整	rem	求余数
fix	向0圆整	round	向靠近整数圆整
floor	向-∞圆整	sign	符号函数
mod	模除求余		

5. 其他函数

在 MATLAB 命令窗口中键入 help datafun，可得部分常用函数。

表 2.6 其他常用函数库

名 称	含 义	名 称	含 义
min	最小值	max	最大值
mean	平均值	median	中位数
std	标准差	diff	相邻元素的差
sort	排序	length	个数
norm	欧氏(Euclidean)长度	sum	总和
prod	总乘积	dot	内积
cumsum	累计元素总和	cumprod	累计元素总乘积
cross	外积	cov	协方差矩阵
gradient	求近似梯度		

2.4 MATLAB 语言中的关系与逻辑运算

用户在编写程序时，经常需要判断数值的大小、某个关系式是否正确，这样就需要用到一些关系运算或逻辑运算符。在执行关系及逻辑运算时，MATLAB 将输入的不为零的数值都视为真(True)，而为零的数值则视为否(False)。被判断为真者的运算输出值用 1 表示，被判断为否者用 0 表示。各种运算必须在两个大小相同的阵列或是矩阵中进行。本节给出 MATLAB 中常用的关系和逻辑运算符。

1. 关系运算

关系运算是用来比较数值关系的运算，表 2.7 给出了 MATLAB 中使用的关系运算符。

表 2.7　MATLAB 中的关系运算符

指　令	含　义	指　令	含　义
<	小于	>=	大于等于
<=	小于等于	==	等于
>	大于	~=	不等于

例如，在 MATLAB 命令窗口中给 a 和 b 两个向量赋值

```
>> a=1:2:15;b=2:1:9;
```

这样 a=[1 3 5 7 9 11 13 15]；b=[2 3 4 5 6 7 8 9]，用户可以点击工作空间中的变量 a 和 b，查看 a 和 b 变量的值。检查 a 中的元素是否大于 b 中的相应元素，可使用下列命令，结果由 ans 变量给出，即

```
>> a>b
ans =
     0     0     1     1     1     1     1     1
```

命令

```
>> a==b
```

是检查 a 中元素是否等于 b 中相应的元素，结果由 ans 变量给出，即

```
ans =
     0     1     0     0     0     0     0     0
```

命令

```
>> a-(b>3)
```

先判断 b 是否大于 3，如果大于，则(b>3)值为 1，否则为 0，然后再用 a 的元素减去(b>3)的值，结果由 ans 变量给出，即

```
ans =
     1     3     4     6     8    10    12    14
```

2. 逻辑运算

MATLAB 中使用的逻辑运算符如表 2.8 所示。

表 2.8　MATLAB 中的逻辑运算符

<table>
<tr><th>指　令</th><th>含　义</th><th>指　令</th><th>含　义</th></tr>
<tr><td>&</td><td>逻辑与 and</td><td>~</td><td>逻辑非 not</td></tr>
<tr><td>|</td><td>逻辑或 or</td><td></td><td></td></tr>
</table>

命令

```
>> (a<2)|(b>6)
```

先判断 a<2 是否成立，如果成立，将(a<2)的值赋为 1，否则为 0；同时判断 b>6 是否成立，如成立，将(b>6)的值赋为 1，否则赋为 0；然后对(a<2)和(b>6)的值进行逻辑或运算。

其结果由 ans 变量给出，即

```
ans =
     1     0     0     0     0     1     1     1
```

3. 逻辑关系函数：

MATLAB 中使用的逻辑关系函数如表 2.9 所示。

表 2.9　MATLAB 中的逻辑关系函数

指　　令	含　　　义
xor	不相同取 1，否则取 0
any	只要有非 0 取 1，否则取 0
all	全为 1 取 1，否则为 0
isnan	为数 NaN 取 1，否则为 0
isinf	为数 inf 取 1，否则为 0
isfinite	有限大小元素取 1，否则为 0
ischar	是字符串取 1，否则为 0
isequal	相等取 1，否则取 0
ismember	两个矩阵是属于关系取 1，否则取 0
isempty	矩阵为空取 1，否则取 0
isletter	是字母取 1，否则取 0(可以是字符串)
isstudent	学生版取 1
1isprime	质数取 1，否则取 0
isreal	实数取 1，否则取 0
isspace	空格位置取 1，否则取 0

命令

```
>> isequal(a,b)
```

判断变量 a 和 b 是否相等。其结果由 ans 变量给出，即

```
ans =
     0
>> isreal(a)
```

此命令判断 a 是否是实数。其结果由 ans 变量给，即

```
ans =
     1
```

习　题

2.1　在 MATLAB 命令窗口中键入数据 a=6.3，记录下列格式设置下，变量 a 的显示值。

format (short)：　短格式(5 位定点数)

format long：　长格式(15 位定点数)

format short e：　短格式 e 方式

format bank：　2 位十进制

2.2 (1) 在 MATLAB 的命令窗口键入 help elfun 和 help datafun，熟悉 MATLAB 常用的函数库。

(2) 令 x=[89.2　83.4　88.5　91.2　84.1　80.4　91.5　92.7　94.0　93.1]，求 x 向量的长度、x 中元素的总和、最大、最小和平均值。

(3) 令 y=0.25 * pi，求 b=ceil(y)，c=fix(y)，d=4 * sin(y) * log(y)的结果。

2.3 在 MATLAB 命令窗口中给 a 和 b 两个向量赋值 a=1:2:15;b=1:1:8;查看 a 和 b 的值，并计算 a-(b＞3)及(a＜2)|(b＞6)的结果，分析为何得出这样的结果。

第三章　数组与矩阵的运算

数组与矩阵是 MATLAB 的基础。MATLAB 事实上是以数组(array)及矩阵(matrix)方式在做运算,而这二者在 MATLAB 中的基本运算性质不同,表现在数组强调元素对元素的运算,而矩阵则采用线性代数的运算方式。

3.1　数组和矩阵的产生

本节介绍在 MATLAB 中如何产生数组和矩阵。由于数组可以定义为只有一行(或一列)的矩阵,因而本节仅讲述矩阵的产生。

3.1.1　逐个元素输入法——直接输入法

MATLAB 的数组与矩阵用中括号[ ]表示。数组由一维元素构成,而矩阵则由多维元素组成。

定义矩阵的原则是:矩阵元素间用空格或逗号隔开,行间用分号隔开。这样程序可以自行解读矩阵的行、列标志和元素。例如

```
>> x=[1 2 3 4 5 6 7 8] ;      % 一维 1x8 数组,逐个元素输入法。
>> x = [1 2 3 4 5 6 7 8; 4 5 6 7 8 9 10 11] ;    % 二维 2x8 矩阵,以分号";"间隔
                                                   各行的元素,逐个元素输入法。
```

3.1.2　快速矩阵产生法

上面的方法只适用于元素不多的情况,元素很多的时候,这种方法就显得很烦琐了。对于多元素的矩阵应采用以下的方式:

(1) 数组的冒号生成法:讲到数组和矩阵的产生,须先介绍一些冒号操作。冒号操作在建立矩阵的索引与引用方面是非常方便的。例如数组的冒号生成法,其生成一维行向量的格式为

x=a:inc:b

其中,a 是数组的第一个元素;inc 是采样点之间的间隔,即步长。若 b−a 是 inc 的整数倍,则所生成数组的最后一个元素等于 b,否则小于 b。inc 可以是正数或负数,亦可以省略,缺省时的默认取值为 1。例如:

```
>> y=1:1:8;      % 一维 1x8 数组,冒号生成法,亦可以是 x=1:8。
```

```
>> y=0:0.02:1; % 以起始值为 0、增量值为 0.02、终止值为 1 的矩阵行向量。
>> x = [1:8; 4:11]
x =
     1     2     3     4     5     6     7     8
     4     5     6     7     8     9    10    11
```

(2) 采用 MATLAB 提供的函数生成矩阵,在 MATLAB 命令窗口键入 help elmat,可以看到 MATLAB 提供的标准矩阵生成函数,见表 3.1。

表 3.1 MATLAB 提供的标准矩阵生成函数

指　令	含　义	指　令	含　义
diag	产生对角形矩矩阵(不适用高维)	rand	产生均匀分布随机数组
eye	产生单位矩阵	randn	产生正态分布随机数组
magic	产生魔方矩阵	zeros	产生全 0 矩阵
ones	产生全 1 矩阵	linspace	在区间上生成线性分度的向量
logspace	在区间上生成对数分度的向量	meshgrid	用于 3D 绘图

例如:>>x=linspace(0,1,101); % 利用 linspace,以产生区间起始值为 0、终止值为 1 之间的元素,元素数目为 101 的矩阵。

```
>>a=[]     %生成空矩阵。
a =
     []
>> zeros(2,3)     %生成 2×3 的全为 0 的矩阵。
ans =
     0     0     0
     0     0     0
>> eye(3)     %生成生成 3×3 的单位阵。
ans =
     1     0     0
     0     1     0
     0     0     1
>>ones(3,3)     %生成全为 1 的矩阵。
ans =
     1     1     1
     1     1     1
     1     1     1
>> rand(2,4);     %生成随机矩阵。
>>a=1:1:10;
```

```
>>b=0.1:0.1:1;
>>c=[b a];        %可利用先前建立的数组 a 及数组 b ,组成新数组。
>>a+b*i           %生成复数数组。
ans =
  Columns 1 through 4
  1.0000 + 0.1000i   2.0000 + 0.2000i   3.0000 + 0.3000i   4.0000 + 0.4000i
  Columns 5 through 8
  5.0000 + 0.5000i   6.0000 + 0.6000i   7.0000 + 0.7000i   8.0000 + 0.8000i
  Columns 9 through 10
   9.0000 + 0.9000i   10.0000 + 1.0000i
>> [X,Y] = meshgrid(-1:.4:1, -0.5:.2:0.5);
>> X
X =
    -1.0000   -0.6000   -0.2000    0.2000    0.6000    1.0000
    -1.0000   -0.6000   -0.2000    0.2000    0.6000    1.0000
    -1.0000   -0.6000   -0.2000    0.2000    0.6000    1.0000
    -1.0000   -0.6000   -0.2000    0.2000    0.6000    1.0000
    -1.0000   -0.6000   -0.2000    0.2000    0.6000    1.0000
    -1.0000   -0.6000   -0.2000    0.2000    0.6000    1.0000
>> Y
Y =
    -0.5000   -0.5000   -0.5000   -0.5000   -0.5000   -0.5000
    -0.3000   -0.3000   -0.3000   -0.3000   -0.3000   -0.3000
    -0.1000   -0.1000   -0.1000   -0.1000   -0.1000   -0.1000
     0.1000    0.1000    0.1000    0.1000    0.1000    0.1000
     0.3000    0.3000    0.3000    0.3000    0.3000    0.3000
     0.5000    0.5000    0.5000    0.5000    0.5000    0.5000
```

说明:meshgrid()函数将第一个行向量扩展成方阵,且各行均是原行向量,结果赋给 X;将第二个行向量也扩展成方阵,且各列均是原行向量的转置,结果赋给 Y。

3.2 矩阵的子矩阵的寻访与赋值

在 MATLAB 的内部资料结构中,每一个矩阵都是一个以行为主(column - oriented)的数组(array),因此对于矩阵元素的存取,可用一维或二维的索引(index)来定址。常用的寻址与赋值指令见表 3.2。

表 3.2　子数组寻址和赋值格式汇总表

子数组寻址和赋值	使　用　说　明
A(r,c)	由 A 的"r 指定行"和"c 指定列"上的元素组成
A(r,:)	由 A 的"r 指定行"和"全部列"上的元素组成
A(:,c)	由 A 的"全部行"和"c 指定列"上的元素组成
A(:)	"单下标全元素"寻访。它由 A 的各列按从左到右的次序，首尾相接而生成"一维长列"数组
A(s)	"单下标"寻访。A(:)中的第 s 个元素
A(r,c)=Sa	"双下标"方式，对子矩阵 A(r,c)赋值，Sa 的"行宽、列长"必须与 A(r,c)的"行宽、列长"相同
A(:)=D(:)	全元素赋值方式。结果：保持 A 的"行宽、列长"。条件是 A 和 D 两个数组的总元素相等，但"行宽、列长"不一定相同。

若 x =[1 4 12 3 6 4 7 5 8 6 9 7 10 8 11]

```
>> x(3)      % x 的第 3 个元素。
ans =
    12
>> x([1 2 5])      % x 的第 1,2,5 个元素。
ans =
    1     4     6
>>  x(1:5)     % x 的第前 5 个元素。
ans =
    1     4     12     3     6
>> x(10:end)     % x 的第 10 个元素后的元素。
ans =
    6     9     7     10     8     11
>> x(10:-1:2)     % x 的第 10 个元素和第 2 个元素的倒排。
ans =
    6     8     5     7     4     6     3     12     4
>> x(find(x>5))     % x 中大于 5 的元素。
ans =
    12     6     7     8     6     9     7     10     8     11

>> x(4)=100     %给 x 的第 4 个元素重新赋值。
x =
  Columns 1 through 12
    1     4     12    100     6     4     7     5     8     6     9     7
```

```
  Columns 13 through 15
10      8      11
>> x(3)=[]     %删除第 3 个元素。
x =
  Columns 1 through 12
     1     4   100     6     4     7     5     8     6     9     7    10
  Columns 13 through 14
     8    11
>> x(16)=1     % 加入第 16 个元素。
x =
Columns 1 through 12
     1     4   100     6     4     7     5     8     6     9     7    10
  Columns 13 through 16
     8    11     0     1
>>x=[1:8;4:11],x(:)=[1:4;2:5;3:6;4:7]
x =
     1     3     2     4     3     5     4     6
     2     4     3     5     4     6     5     7
```

3.3 矩阵的运算

3.3.1 矩阵的算术运算

矩阵的几种算术运算见表 3.3。

表 3.3 矩阵的算术运算符

算术运算	运算符	MATLAB 表达式	使用说明
加	+	A+B	表示 A 与 B 相加
减	−	A−B	表示 A 与 B 相减
乘	*	A*B	表示 A 与 B 相乘
除	/或\	A/B	表示 A 的逆与 B 的左乘，即 inv(A) *B
		A\B	表示 A 的逆与 B 的右乘，即 B*inv(A)
幂	^	A^B	表示对 A 求 B 次幂

3.3.2 矩阵的转置

转置是一种重要的矩阵运算，通过在矩阵变量后加"'"的方法来表示转置运算。例如：

```
>> A=[1 2 3;4 5 6]
A =
     1     2     3
     4     5     6
>> B=A'
B =
     1     4
     2     5
     3     6
```

3.3.3 元素群的运算(点运算)

在前几节所讲的常见运算中,各种操作都是针对矩阵的所有元素或一部分元素的操作。MATLAB 中可以对矩阵元素进行单独的操作运算。对于加减法,对矩阵元素的操作和对矩阵的操作是一致的。对于其他运算,在对矩阵元素进行操作时均需要在操作符前加点。如

.*   ./   .^

元素群的运算是各变量中对应元素的运算。例如:

```
>> A=[1 2; -1 5];B=[6 3; 1 0];
>> A.*B    %矩阵对应元素相乘。
ans =
     6     6
    -1     0
>> B./A    %矩阵对应元素相除。
ans =
    6.0000    1.5000
   -1.0000    0
>> B.^2    %矩阵元素乘方运算。
ans =
    36     9
     1     0
```

3.4 矩阵函数

3.4.1 元素群的函数(elfun 基本函数库)

2.5 节中给出了 MATLAB 常用数学函数,这些数学函数一般都可以作为矩阵函数。例如:

```
>> A=1:1:5;B=0:10:50;
```

```
>> sin(A)   %对矩阵 A 中各元素求正弦函数值。
ans =
   0.8415    0.9093    0.1411   -0.7568   -0.9589
>> sign(A)
ans =
    1     1     1     1     1
>> mean(B)
ans =
   25
>> C=[1 2;2 5];D=exp(C)
D =
   2.7183     7.3891
   7.3891   148.4132
>> size(D)       %求矩阵 D 的维数。
ans =
    2     2
>> length(B)       %求向量 B 的长度。
ans =
    6
```

3.4.2　常用矩阵变换函数

在 MATLAB 命令窗口键入 help elmat，可得常用的矩阵变换函数，如表 3.4 所示。

表 3.4　常用的矩阵变换函数

名　称	含　　义	名　称	含　　义
fliplr	矩阵左右翻转	diag	产生或提取对角矩阵
flipud	矩阵上下翻转	tril	产生下三角矩阵
flipdim	矩阵特定维翻转	triu	产生上三角矩阵
Rot90	矩阵反时针 90°翻转	find	查找非零元素
End	向量的最后元素	reshape	矩阵重组

例如：

```
>> A=[10,2,12;34,2,4;98,34,6];
>> fliplr(A)
ans =
   12     2    10
    4     2    34
    6    34    98
>> flipud(A)
```

```
ans =
    98    34     6
    34     2     4
    10     2    12
>> rot90(A)
ans =
    12     4     6
     2     2    34
    10    34    98
>> tril(A)
ans =
    10     0     0
    34     2     0
    98    34     6
>> triu(A)
ans =
    10     2    12
     0     2     4
     0     0     6
```

3.4.3 矩阵的专用函数

MATLAB 中还提供了一些关于矩阵的专用函数，用户只需在 MATLAB 命令窗口键入 help matfun，即可看到如表 3.5 所示函数的在线帮助。

表 3.5 矩阵的专用函数

名 称	含 义	名 称	含 义
norm	计算矩阵范数	inv	计算矩阵的逆
rank	计算矩阵的秩	cond	计算矩阵的条件数
det	计算矩阵的行列式值	chol	Cholesky 分解
trace	计算矩阵的迹	lu	Lu 分解
null	计算矩阵的零空间	pinv	计算矩阵的伪逆
eig	计算矩阵的特征值、特征向量	svd	奇异值分解
poly	计算矩阵的特征多项式	expm	矩阵的指数函数
logm	矩阵的对数函数	sqrtm	矩阵的开方根函数
funm	一般矩阵函数		

例如：

```
>> A=[10,2,12;34,2,4;98,34,6];
```

```
>> rank(A)
ans =
     3
>> det(A)
ans =
     10656
>> inv(A)
ans =
   -0.0116     0.0372    -0.0015
    0.0176    -0.1047     0.0345
    0.0901    -0.0135    -0.0045
>> [v,u]=eig(A)        %求A矩阵的特征值和特征向量,返回值中v为特征向量,u是
                       特征值。
v =
   -0.2960    -0.3635     0.3600
   -0.2925     0.4128    -0.7886
   -0.9093     0.8352    -0.4985
u =
   48.8395          0          0
         0   -19.8451          0
         0          0   -10.9943
>> C=eye(2);expm(C)
ans =
    2.7183          0
    0          2.7183
>> F = funm(A,@sin)     %一般矩阵函数,对A矩阵求正弦。
F =
   -1.2022    -1.3169     0.4929
    0.5776     1.9880    -1.1459
    0.1975     1.7427    -1.6144
>> norm(A)
ans =
  109.5895
>> norm(A,1)
ans =
   142
>> norm(A,inf)
ans =
   138
```

习　题

3.1　用 clear 清除 MATLAB 工作空间中的所有变量后，构造 **A**,**B**,**C**,**D** 矩阵分别是 3×3 维单位矩阵、3×2 维全零矩阵、1×3 维全 1 矩阵及 1×2 维全零矩阵，然后求 **F**=[**A B**;**C D**]的结果。

3.2　在 MATLAB 命令窗口中键入 help meshgrid，学习该函数的使用方法，求[X,Y] = meshgrid(−2:.8:2, −0.5:.2:0.5)的结果。

3.3　矩阵的子矩阵的寻访与赋值练习。

若 x =[1 : 8 , 10 : 16]，在 MATLAB 命令窗口键入 x(3)求 x 的第 3 个元素；键入 x([1 2 5])求 x 的第 1,2,5 个元素；键入 x(1:5)求 x 的前 5 个元素；键入 x(10:−1:2)求 x 的第 10 个元素和第 2 个元素的倒排；x(4)=100 给第 4 个元素重新赋值，观察 x 值的变化情况。

3.4　用 clear 清除 MATLAB 工作空间中的所有变量后，令 x=0:0.05:1；y=0:0.1:2；求 z=sin(x * pi).* cos(y * pi)的值。

3.5　求下列矩阵的行列式、逆阵、迹、秩、特征多项式、特征值、特征向量。

$$\boldsymbol{A}=\begin{bmatrix}7.5 & 3.5 & 0 & 0\\ 8 & 33 & 4.1 & 0\\ 0 & 9 & 103 & -1.5\\ 0 & 0 & 3.7 & 19.3\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}5 & 7 & 6 & 5\\ 7 & 10 & 8 & 7\\ 6 & 8 & 10 & 9\\ 5 & 7 & 9 & 10\end{bmatrix},\quad \boldsymbol{C}=\begin{bmatrix}1 & 2 & 3 & 4\\ 5 & 6 & 7 & 8\\ 9 & 10 & 11 & 12\\ 13 & 14 & 15 & 16\end{bmatrix}$$

3.6　A=[1 2 3;4 5 6;7 8 9]；A11=[A zeros(3,1);[1 3 5] 7]；求 alr=fliplr(A11)、aud=flipud(A11)和 a90=rot90(A11)。

第四章　多项式的表达式及其操作

控制系统的传递函数常常用分子、分母均为多项式的分数表示，MATLAB 提供了标准多项式运算的函数，如多项式求根、求值和微分。

4.1　多项式的表达和生成

4.1.1　多项式表达式的约定

MATLAB 约定降幂多项式 $P(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$ 用系数行向量 $\boldsymbol{P}=[a_n\ \ a_{n-1}\ \ \cdots\ \ a_1\ \ a_0]$表示。

4.1.2　多项式行向量的生成方法

多项式行向量的生成方法有两种，分别是直接输入法和利用指令生成法。

1. 直接输入法

按照约定，将多项式的各项系数依降幂次序排放在行向量的元素位置上。要注意的是，多项式中缺的幂次项的系数记为零。

2. 利用指令生成法

利用指令 P=poly(AR)生成多项式系数向量。其中，若 AR 是方阵，则多项式 P 就是该方阵的特征多项式；若 AR 是行向量，即 AR=$[ar_1\ \ ar_2\ \ \cdots\ \ ar_n]$，则 AR 的元素被认为是多项式 P 的根，即所得的多项式满足关系式：$p=(x-ar_1)(x-ar_2)\cdots(x-ar_n)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$。

得到了多项式的向量表达式 P 后，利用命令 poly2str(P,'s')可以得到习惯方式显示的多项式，其中，s 是多项式中的自变量。例如：

```
>> AR=[1 4 7;3 11 6;5 32 68];
>> P=poly(AR)      %AR 的特征多项式。
P =
    1.0000   -80.0000   588.0000 -147.0000
>> PPA=poly2str(P,'s')      %用习惯的方式显示多项式。
PPA =
   s^3 - 80 s^2 + 588 s - 147
```

再例：

```
>> R=[-0.5 -0.3+0.4i -0.3-0.4i]      %根向量。
R =
  -0.5000    -0.3000 + 0.4000i    -0.3000 - 0.4000i
>> P=poly(R)      % R的特征多项式。
P =
    1.0000      1.1000      0.5500      0.1250
>> PPR=poly2str(P,'x')      %用习惯的方式显示多项式。
PPR =
   x^3 + 1.1 x^2 + 0.55 x + 0.125
```

4.2 多项式运算函数

表 4.1 给出了 MATLAB 中多项式运算函数及其调用格式，利用这些函数，可以对多项式进行运算。

表 4.1　多项式运算函数的调用格式

指　　令	说　　明
p=conv(p1,p2)	p 是多项式 p1 和 p2 的乘积多项式
[q,r]=deconv(p1,p2)	多项式 p1 被 p2 除的商多项式为 q，余多项式是 r
p=poly(AR)	计算方阵 AR 的特征多项式 p；或求向量 AR 指定根对应的多项式
dp=polyder(p)	计算多项式 p 的导数多项式 dp
dp=polyder(p1,p2)	计算多项式 p1 和 p2 乘积的导数多项式 dp
[num,den]= polyder(p1,p2)	对有理分式(p1/p2)求导，得有理分式(num/den)
p=polyfit(x,y,n)	求 x,y 向量给定的数组的 n 阶拟合多项式 p
pA=polyval(p,S)	按数组运算规则计算多项式值。p 为多项式，S 为矩阵
pM=polyval(p,S)	按矩阵运算规则计算多项式值。p 为多项式，S 为矩阵
[r,p,k]=residue(b,a)	部分分式展开：b,a 分别是分子、分母多项式系数向量；r,p,k 分别是留数、极点和直项
r=roots(p)	计算多项式 p 的根

例 4.1　求 $\frac{(s^2+2)(s+4)(s+1)}{s^3+s+1}$ 的“商”和“余”多项式。

```
>> p1=conv([1 0 2],conv([1 4],[1 1]));   %计算分子多项式。
>> p2=[1 0 1 1];
>> [q,r]=deconv(p1,p2);
```

```
>> cq='商多项式为';cr='余多项式为';
>> disp([cq,poly2str(q,'s')]),disp([cr,poly2str(r,'s')])
商多项式为    s+5
余多项式为    5s^2+4s+ 3
```

例 4.2　将 $\frac{1}{s^2+5s+6}$ 展开成部分分式，结果应为$\frac{1}{s+2}-\frac{1}{s+3}$。

```
>>a=[1 5 6];b=[1];
>>[r,s,k]=residue(b,a)
r =
   -1.0000
    1.0000
s =
   -3.0000
   -2.0000
k =
[]
```

说明：当分母多项式阶次高于分子多项式阶次时，k 的计算结果是空矩阵，表示无此项。

例 4.3　求根。

```
>> p=[1 3 6 20 69];roots(p)
ans =
  -2.5473 + 1.5948i
  -2.5473 - 1.5948i
   1.0473 + 2.5578i
   1.0473 - 2.5578i
```

习　题

4.1　多项式及多项式函数练习：令 a=[1 2 3;4 5 6;7 8 9];p=[3 0 2 3];q=[2 3],x=2，求 roots(p),p1=conv(p,q),p2=poly(a),p3=polyder(p),p4=polyval(p,x)的结果。

4.2　将 $\frac{1}{s^2+3s+2}$ 展开成部分分式。

4.3　求 $\frac{(s^2+2)(s+4)}{s^2+s+1}$ 的商和余数。

第五章　字符串和其他数据类型

前面介绍了 MATLAB 中常用数值数组(Numeric Array)。本章将介绍其他几种 MATLAB 数据类型：字符串类型(Character String Array)、元胞数组(Cell Array)和结构体数组(Structure Array)。

5.1 字符串及其处理

5.1.1 字符串的基本操作

在 MATLAB 工作空间中，字符串是以向量形式来存储的，所以可以通过它的下标对字符串中的任何一个元素进行访问。MATLAB 中的字符串用一对单引号来定义，例如：

```
>>s='student'
s =
student
```

这样，工作空间中就有一个字符串变量，变量名称 s，变量值为'student'，变量类型 char。

字符串可以像数据矩阵一样来连接。用于数据矩阵的求长度和大小的函数 length()和 size()，亦可以用来求字符串的长度和大小。而且，冒号表达式的使用和在数据矩阵中的使用情况完全相同。例如：

```
>> name='Mary';s='student';
>> s1=[name s]
s1 =
    Marystudent
>> s3=[name blanks(3);s]     % 各行元素的列数必须相同，故使用 blanks()函数第一行补了三个空格，blanks()函数的用法详见 5.1.2 小节。
s3 =
Mary
student
```

再如：>> s2=[name, ' is a good ',s,'.']

```
s2 =
```

```
Mary is a good student.
>> length(s1)    %求字符串的长度。
ans =
    11
>> size(s3)
ans =
     2     7
>> name1=s1(1:5)
name1 =
Marys
```

注意：长字符串可以采用分段表示的方法。例如：

```
>> keep={...
'I went to work by bicycle,'
'while my students went to school by foot'}
keep =
    'I went to work by bicycle,'
    'while my students went to school by foot'
```

5.1.2 字符串函数

在 MATLAB 命令窗口中键入 help strfun，可以得到一些转换函数和操作函数，分别见表 5.1 和表 5.2。

表 5.1　字符串转换函数

指　令	含　　义	指　令	含　　义
abs	把字符串翻译成 ASCII 码	hex2dec	十六进制字符串转换为十进制整数
bin2dec	二进制字符串转换为十进制整数	hex2num	十六进制字符串转换为浮点数
str2num	把字符串转换为数值	int2str	整数转换为字符串
dec2base	十进制整数转换为 X 进制字符串	mat2str	把数值矩阵转换为 eval 可调用的格式
dec2bin	十进制整数转换为二进制字符串	num2str	把数值转换为字符串
dec2hex	十进制整数转换为十六进制字符串	setstr	把 ASCII 码翻译为字符串
double	把任何类型数据转换为双精度数值	sprintf	以控制格式把数值转化为字符串
fprintf	把格式化数据写到文件或屏幕	sscanf	以控制格式把串转化为数
str2double	把字符串转换为双精度数值		

举例说明 fprintf，sprintf，sscanf 的用法。

```
>> rand('state',0);a=rand(2,2);     %产生 2×2 随机矩阵。
>> ss=sprintf('%.10e\n',a)      %以 10 位数科学计数字符串显示，且每写一个元素
                                换一行。
```

```
ss =
9.5012928515e-001
2.3113851357e-001
6.0684258354e-001
4.8598246871e-001
>> fprintf('%.5g\\',a)    %以 5 位数位最短形式显示。
0.95013\0.23114\0.60684\0.48598\
>> ssca=sscanf(ss,'%f',[3,2])    %以浮点格式把字符串转换成(3×2)矩阵。
ssca =
    0.9501    0.4860
    0.2311         0
    0.6068         0
>> A=[1 1 3;2 4 7];mat2str(A)
ans =
    [1 1 3;2 4 7]
```

表 5.2 字符串操作函数

指 令	含 义	指 令	含 义
blanks(n)	创建 n 个空格字符串	lower(s)	使 s 里的英文字母全部小写
char(s1,s2,…)	把字符串 s1,s2 等逐个写成行，形成多行数值	str2mat(s1,s2,…)	把字符串 s1,s2 等逐个写成行，形成多行数值，并删去全空行
deblank(s)	删去字符串尾部的空格符	strcat(s1,s2,…)	把串 s1,s2 等连接成长字符串
eval(s)	把字符串 s 当做 MATLAB 指令运行	strcmp(s1,s2)	若字符串 s1,s2 相同，则判为'真'，给出逻辑 1
eval(s1,sc)	把字符串 s1 当做 MATLAB 指令运行，如 s1 发生错误，则运行 sc	strjust(s)	字符串的对齐方式：右对齐、左对齐或对中
feval(f,x,y,…)	对输入量 x,y 等计算函数 f	strmatch(s1,s2)	逐行字符串 s2，给出以字符串 s1 开头的行的行号
findstr(s1,s2)	在较长串中，找出短字符串的起始字符的下标	strncmp(s1,s2,n)	若字符串 s1,s2 的前 n 个字符相同，则判为'真'，给出逻辑 1
ischar(s)	s 是字符字符串，则判为'真'，给出逻辑 1	strrep(s1,s2,s3)	将字符串 s1 中所有出现 s2 的地方替换为 s3
isletter(s)	以逻辑 1 指示 s 里文字符的位置	strtok(s)	找出第一个间隔符(空格、制表位、回车符)前的内容
isspace(s)	以逻辑 1 指示 s 里空格符的位置	strvcat(s1,s2,…)	把字符串 s1,s2 等逐个写成行，形成多行数值
lasterr	MATLAB 发出的最新错误信息	upper(s)	使 s 里的英文字母全部大写

例如：

```
>> name=upper('matlab')
name =
MATLAB
>> fun=strrep('hahaha','a','i')
fun =
hihihi
> text=...
'Monday,Tuesday,Wednsday,...';
>> [day,rest]=strtok(text,',')
day =
Monday
rest =
,Tuesday,Wednsday,...
>> [day,rest]=strtok(text,'a')        %找出第一个a字母之前的内容。
day =
Mond
rest =
ay,Tuesday,Wednsday,...
>> A=[1 1 3;2 4 7];text1='ResultMatrix=';text2=mat2str(A);
>> restext=strcat(text1,text2)
restext =
ResultMatrix=[1 1 3;2 4 7]
```

5.2 元胞数组

前面所提到的矩阵和数组中，所有元素的数据类型都是相同的。MATLAB 中的元胞数组(亦称之为元胞或细胞数组等)支持复合数据类型的矩阵和数组。其内部不同的位置可以有不同的数据类型。这是它与由数据或字符构成的普通数组的不同之处。元胞数组可以是一维的，也可以是多维的，其引用方式和普通数组类似，但其生成方式和普通数组的生成方式是不相同。

5.2.1 元胞数组的生成和显示

对于元胞数组来说，元胞和元胞内的内容是两个不同范畴的东西。因此寻访元胞和寻访元胞中的内容是两种不同的操作。因此，MATLAB 设计了两种不同的操作：元胞外标识和元胞内编址。对外标识元胞元素的操作采用“圆括号”；而对编址元胞元素内涵的操作采用“花括号”。例如：

```
>> C_str=char('这是','元胞数组生成显示算例');  %生成字符串。
>> R=reshape(1:9,3,3);  %生成(3×3)实矩阵。
>> S_sym=sym('sin(4 * t) * exp(-2 * t)');  %生成符号函数量。
>> Cn=[1+2i];  % 生成复数。
```

外标识元胞元素赋值法：

```
>> A(1,1)={C_str};A(1,2)={R};A(2,1)={Cn};A(2,2)={S_sym};
>> A      %显示元胞数组。
A =
          [2x10 char]     [3x3 double]
    [1.0000+ 2.0000i]     [1x1 sym]
```

编址元胞元素内涵的直接赋值法：

```
>> B{1,1}=C_str;B{1,2}=R;B{2,1}=Cn;B{2,2}=S_sym;
>> celldisp(B)       %显示 B，celldisp 是显示元胞全部或部分内容的指令。
 B{1,1} =
这是
元胞数组生成显示算例
 B{2,1} =
    1.0000 + 2.0000i
B{1,2} =
      1     4     7
      2     5     8
      3     6     9
B{2,2} =
 sin(4 * t) * exp(-2 * t)
```

5.2.2 元胞数组的扩充、收缩和重组

元胞数组的扩充、收缩和重组的方法大致与普通数组的情况相同。

1. *元胞数组的扩充*

例如：

```
>> C=cell(2);   %用 cell 指令预设(2×2)空元胞数组。
>> C(:,1)={char('Another','Example');10:-1:1}   %对第一列元胞赋值。
C =
    [2x7    char   ]      []
    [1x10 double]         []
>> AC=[A C],A_C=[A;C]     % 元胞数组的行、列扩充。
AC =
          [2x10 char]     [3x3 double]     [2x7    char   ]     []
    [1.0000+ 2.0000i]     [1x1 sym]        [1x10 double]        []
A_C =
```

```
        [2x10 char   ]    [3x3 double]
    [1.0000+ 2.0000i]    [1x1 sym]
        [2x7    char   ]              []
        [1x10 double]                 []
```

2. 元胞数组的收缩

例如：

```
>> A_C(3,:)=[]    % 删除第三行，使 A_C 成为(3×2)的元胞数组。
A_C =
        [2x10 char   ]    [3x3 double]
    [1.0000+ 2.0000i]    [1x1 sym    ]
        [1x10 double]                []
```

3. 元胞数组的重组

例如：

```
>> R_A_C=reshape(A_C,2,3)
R_A_C =
        [2x10 char]             [1x10 double]    [1x1 sym]
        [1.0000+ 2.0000i]    [3x3    double]         []
```

5.2.3　元胞数组内容的调用

例如：

```
>> f1=R_A_C(1,3)    %使用圆括号寻访得到的是元胞。
f1 =
    [1x1 sym]
>> class(f1)
ans =
cell
```

例如：

```
>> f2=R_A_C{1,3}      %使用花括号寻访得到的是元胞内容。
 f2 =
 sin(4 * t) * exp(-2 * t)
 >> class(f2)
ans =
sym
```

例如：

```
>> f3=R_A_C{1,1}(:,[1 2 5 6])   %取元胞内的子数组。
f3 =
这是
元胞生成
```

例如：

```
>> [f4,f5,f6]=deal(R_A_C{[1,3,4]})    % 同时调用多个元胞内容。
f4 =
这是
单元数组生成显示算例
f5 =
    10     9     8     7     6     5     4     3     2     1
f6 =
     1     4     7
     3     6     9
```

5.3　结构体数组

当今的程序设计语言中大都支持结构体变量，MATLAB 也支持结构体变量。与元胞数组一样，结构体数组也可以在一个数组中存放各种数据类型，它是某些具有某种相关性记录的集合体，它使一系列相关记录集合到一个统一的结构之中，从而使这些记录能够被有效地管理、组织和调用。从某种意义上讲，结构体数组组织数据的能力比元胞数组更强，且其生成与使用都非常容易、直观。

在 MATLAB 中，结构体是按照域的方式生成与存储其中的每个记录的。一个域中可以包含任何 MATLAB 支持的数据类型，如双精度数值、字符、元胞数组及结构体等类型。

5.3.1　结构体数组的生成与显示

结构体生成方式：struct_name(record_number). field_name=data;

例如某个班级学生花名册的建立：

```
>> student(1).name='Li li';
>> student(1).number='609510';
>> student(2).name='Wang Hong';
>> student(2).number='609511';
>> student(3).name='Zhang Liang';
>> student(3).number='609512';
>> student(4).name='Zhao Qi';
>> student(4).number='609513';
```

student 是具有 4 个结构变量的向量，表示某个班级共有 4 个同学的姓名和学号。

```
>> student      %显示结构体。
student =
1x4 struct array with fields:
    name
    number
```

5.3.2 结构体数组的调用和设置

由于结构体数组的域是存放数据的场所，因此，在调用结构体和设置结构体数组中数据的前提是"事先知道域名"。用户可以直接在MATLAB命令窗口中键入结构体名称来查找域名，亦可以采用提供的fieldnames指令来获得域名。一旦有了域名，MATLAB就可以利用getfield和setfield指令实现对结构体数据的调用和设置。

fieldnames，getfield和setfield指令的基本使用格式是：

```
FN=fieldnames(S_n)          %获得结构体域名。
FC=getfield(S_n,{S_index},f_name,{f_index})     %获得结构体域中的内容。
S_n=setfield(S_n,{S_index},f_name,{f_index},value)     %设置结构体域中的内容。
```

例
```
>> FN=fieldnames(student)     % 获得结构体的域名。
FN =
    'name'
    'number'
```

例如：结构体变量的调用可以采用两种方式。简单的方法是

```
>> student(2).name
ans =
Wang Hong
```

亦可采用MATLAB提供的getfield指令，针对本例，有

```
>> FC=getfield(student,{2},'name')     %获得结构体中第二条记录，域名为'name'的内容。
FC =
    Wang Hong
```

例如：对结构体数据的设置需采用setfield指令，如

```
>> student=setfield(student,{2},'name','Li wei')    %将结构体中第二条记录域名为'name'的内容改为'Li wei'。
student =
1x4 struct array with fields:
    name
    number
>> FC=getfield(student,{2},'name')   %显示结构体中第二条记录域名为'name'的内容已更改。
FC =
li wei
```

5.3.3 结构体数组的扩充与收缩

```
>> student(5).number='609514';
>> student(5).name='han yi';
>> student
```

```
student =
1x5 struct array with fields: %扩充为(1×5)的结构体。
    name
number
>> student(:,4)=[]          %又收缩为(1×4)的结构体。
student =
1x4 struct array with fields:
    name
number
```

5.3.4 域的增添和删除

1. 增添域操作

例如：

>> student(1).Average_Score=79.8;student(1).List_number=3; %在数组中的任意记录上进行域的增添操作，其影响遍及整个结构体。

```
>> student
student =
1x4 struct array with fields:
    name
    number
    Average_Score
    List_number
```

2. 删除域操作

用户必须使用 refield 命令对结构体的域进行删减。例如：

```
>> student=rmfield(student,'List_number')
student =
1x4 struct array with fields:
    name
    number
    Average_Score
```

习 题

5.1 图形标注中较多用到数据到字符串的转换函数 int2str，num2str 和 mat2str。

(1) 在 MATLAB 命令窗口中生成一个整数数组，如使用命令 eye(2,4)，并将其转换为字符串数组；

(2) 将数据 1 234.5 转换为字符串；

(3) 在 MATLAB 命令窗口中生成一个随机的 2×4 矩阵，使用 mat2str 命令将其转换成输入形态的串数组，使用函数 rand 生成随机数矩阵。

5.2 将字符串 MATLAB 改写为小写 matlab 字符串。

5.3 将字符串 a='my first' ,b='examp'串在一行显示。

第六章　MATLAB 基本绘图功能

在进行科学分析时，人们常常需要将所得到的数据进行分析，又为了能够直观地看出数据的趋势，用户一般需要将数据绘制成曲线。本章介绍 MATLAB 二维图形、三维图形及特殊图形的绘图命令。

6.1　二维图形的绘制

6.1.1　基本绘图函数

MATLAB 中常用二维图形的基本绘图函数是 plot，除此以外，还有一些绘制对数坐标图的绘图函数，见表 6.1。

表 6.1　基本绘图函数

命　　令	含　　　义
plot	建立向量或矩阵各队向量的图形
loglog	x，y 轴都取对数标度建立图形
semilogx	x 轴用对数标度，y 轴用线性标度绘制图形
semilogy	y 轴用对数标度，x 轴用线性标度绘制图形
plotyy	在图的左、右两侧分别建立纵坐标轴

6.1.2　绘制二维图形的一般步骤

MATLAB 中二维图形的绘制一般包括数据准备、选择图形窗口、选择子图位置、调用绘图函数、设定坐标轴范围、刻度，并进行图形注释等。表 6.2 给出了各部分具体实现所需使用的指令。

表 6.2 绘制二维图形的一般步骤

序号	步 骤	典 型 指 令
1	数据准备：包括选定所要表现的范围；生成自变量采样向量；计算相应的函数值向量	x=(0:pi/50:2*pi)'; y=sin(x).*sin(9*x);
2	选择图形窗口和子图的位置	figure(2) %指定 2 号图形窗口 subplot(2,2,3) %将当前图形窗口分为 2 行 2 列 4 个子图，指定 3 号子图
3	调用绘图指令：规定线型、色彩、数据点型	plot(x,y,'b—') %用蓝色实线画曲线
4	设置坐标轴的范围、刻度和分格线	axis([0,pi,-1,1]) %设置坐标轴的范围 grid on %绘制坐标分格线
5	图形注释：设置图名、坐标名、图例、文字说明	title('数字仿真结果') %图名 xlabel('x');ylabel('y') %坐标名 legend('y=sin(x)*sin(9*x)') %图例 text(2,0.5,' y=sin(x)*sin(9*x)') %文字说明

6.1.3 二维图形绘图的基本操作

1. plot 的基本调用格式

plot(x,y)：输出以向量 x 为横坐标，向量 y 为纵坐标的图形，x、y 必须具有相同的长度。

plot(y)：输出以向量 y 元素序号 m 为横坐标，向量 y 对应元素 m 为纵坐标的图形。

plot(x,y,'str')：用'str'指定的方式，输出以 x 为横坐标，y 为纵坐标的图形。在指定方式 str 中，用户可以规定绘制曲线的线型、数据点型、颜色等。

plot(x1,y1,'str1',x2,y2,'str2',…)：在一副图中，用'str1'指定的方式，输出以 x1 为横坐标，y1 为纵坐标的图形。用'str2'指定的方式，输出以 x2 为横坐标，y2 为纵坐标的图形。'str'选项中的部分参数如表 6.3 所示。表 6.3 中定义了曲线线型、色彩、数据点型的允许值，它们可以单独使用，也可以组合使用。下面通过例题说明 plot 指令的用法。

表 6.3 曲线线型、色彩、数据点型允许设置值

颜 色		图 线 型 态			
字元	颜 色	字元	数据点型	字元	线 型
y	黄 色	.	点	-.	点虚线
k	黑 色	o	圆	--	虚 线
w	白 色	x	x	-	实 线
b	蓝 色	+	+	:	点 线
g	绿 色	*	*		
r	红 色	d	菱 形		
c	亮青色	^	上三角		
m	锰紫色	p	五角星		
		h	六角星		
		s	方 形		

例 6.1 在 MATLAB 命令窗口中键入如下命令

```
>> t=(0:pi/50:2*pi)';
>> k=0.4:0.1:1;
>> Y=cos(t)*k;
>> plot(t,Y)                          %绘图结果见图 6.1。
```

例 6.2

```
>> t=(0:pi/100:pi)';                  %准备数据。
>> y1=sin(t)*[1,-1];
>> y2=sin(t).*sin(9*t);
>> t3=pi*(0:9)/9;
>> y3=sin(t3).*sin(9*t3);
>> plot(t,y1,'b:',t,y2,'k-',t3,y3,'ro')  %绘图结果见图 6.2。
>> axis([0 pi -1 1])                  %控制坐标轴范围。
```

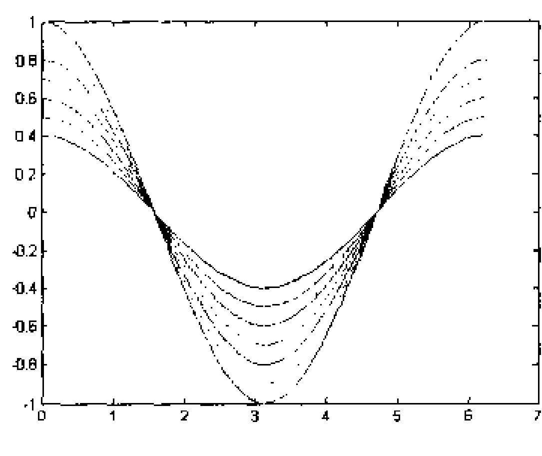


图 6.1 plot 指令基本操作演示 1

图 6.2 plot 指令基本操作演示 2

除了利用指令设置线型、色彩和数据点型等属性外，用户还可以在图形窗口中，激活 edit plot，用左键双击需要设置的曲线，即可打开曲线属性编辑器（见图 6.3）设置曲线属性。感兴趣的读者可以试试。

2. 坐标、刻度和分格线控制

(1) 坐标控制：人工设置坐标范围的指令是：

axis(V)

其中，设置二维坐标系范围时，取 V=[x1, x2, y1,y2]；而设置三维坐标系范围时，取 V=[x1, x2, y1,y2,z1,z2]。V 中的元素必须满足条件：x1<x2, y1<y2,z1<z2。

除了指令设置坐标属性外，用户还可以在图形窗口中，激活 edit plot，用鼠标左键双击图形中的任意空白处，即可打开坐标的属性编辑器（见图 6.4），设置坐标属性。感兴趣的读者不妨试试利用坐标属性编辑器进行坐标范围设置。

(2) 刻度设置：MATLAB 允许用户利用对象图柄指令设置坐标刻度。

二维坐标刻度设置：

set(gca,'Xtick',xs,'Ytick',ys)

三维坐标刻度设置：

set(gca,'X'tick',xs,'Ytick',ys,'Ztick',zs)

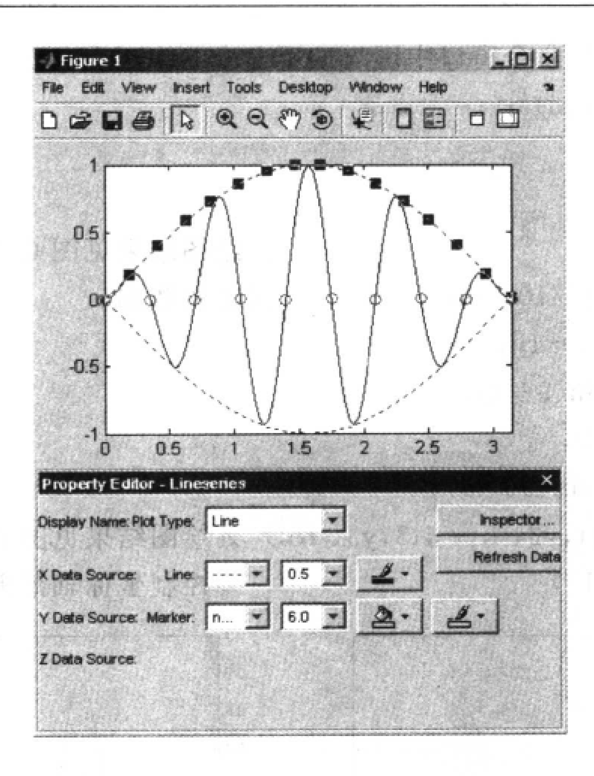


图 6.3 曲线属性编辑器

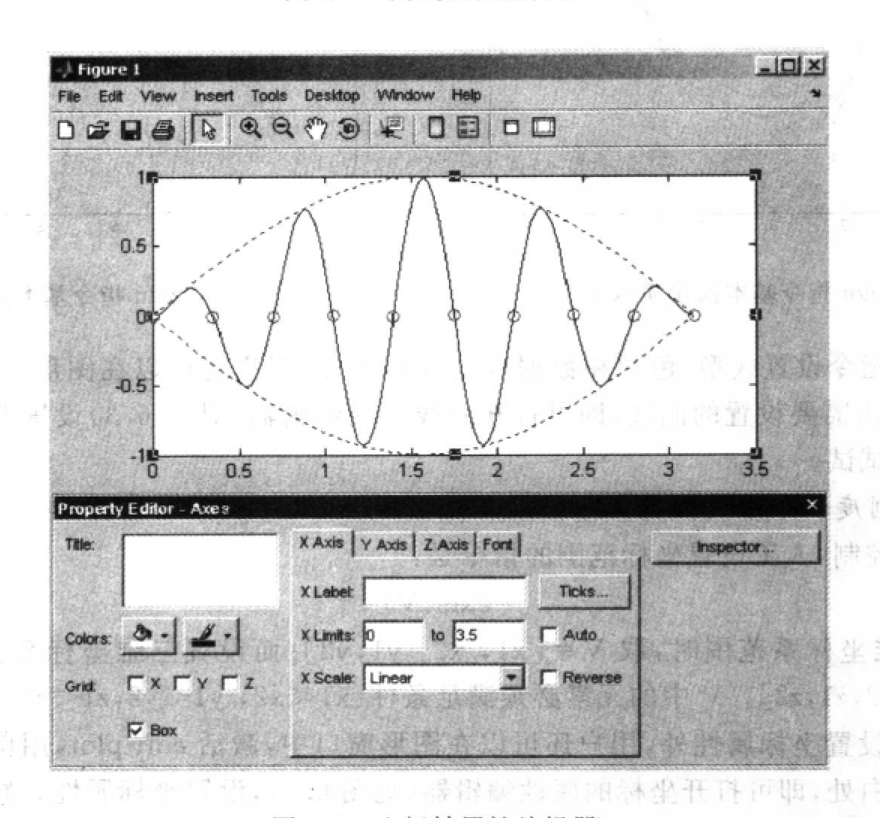


图 6.4 坐标轴属性编辑器

其中，xs，ys，zs 分别决定 x，y 和 z 轴的刻度，可以是任何合法的实数向量。

若在例 6.2 的程序最后加入下列指令，则所绘制的曲线的横纵轴的刻度发生变化，结果如图 6.5 所示。

```
>> set(gca,'Xtick',[0:9]/9*pi,'Ytick',[0.7 0.8 0.9 max(y2)])        %设置刻度。
```

(3) 分格线:

grid　　　　是否画分格线的双向切换指令。

grid on　　　画出分格线。

grid off　　　不画出分格线。

不画分格线是 MATLAB 的缺省设置。分格线的疏密取决于坐标刻度,如想改变分格线的疏密,必须先定义坐标刻度。

(4) 坐标框:

box　　　　坐标形式在封闭式和开启式之间切换指令。

box on　　　使当前坐标呈封闭形式。

box off　　　使当前坐标呈开启形式。

其中封闭形式的坐标是 MATLAB 的缺省设置。

如在例 6.2 中,最后再加入如下两条指令,则绘图结果如图 6.6 所示。

```
>> grid on          %画出分格线。
>> box off          %坐标呈开启形式。
```

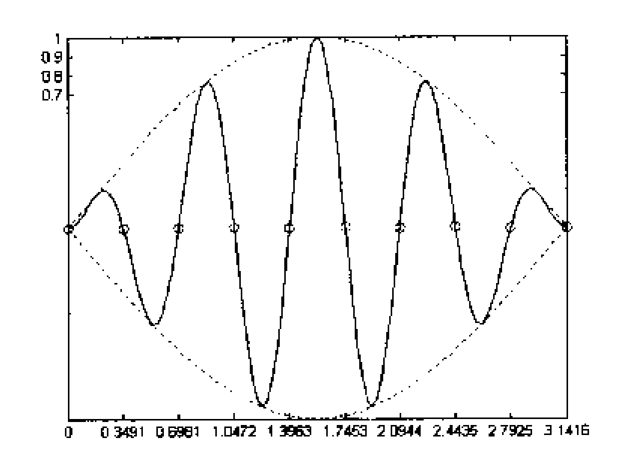


图 6.5　刻度设置结果

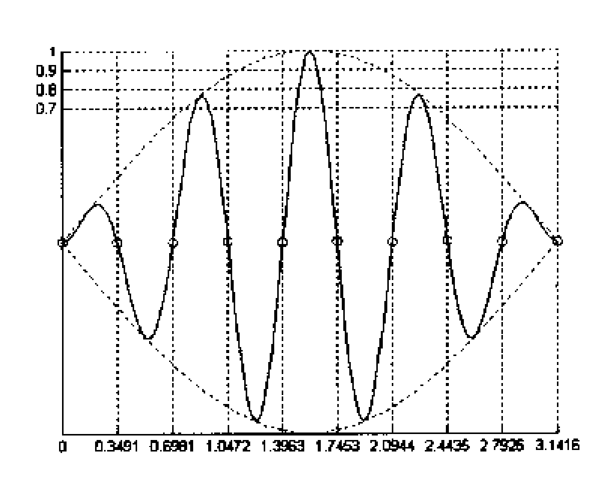


图 6.6　加入分格线、坐标呈开启形式

3. 图形标识

MATLAB 提供的图形标注的命令见表 6.4。

表 6.4　常用图形标注命令

命　令	含　　义
title	给图形加标题
xlabel	给 x 轴加标记
ylabel	给 y 轴加标记
zlabel	给 z 轴加标记
text	在图形指定的位置上加文本字符串
gtext	在鼠标的位置上加文本字符串
legend	给图形加注解

下面举例说明各种图形标注命令的调用格式。标注结果如图 6.7 所示。

例 6.3 各种图形标注命令说明。

```
>> x=-pi:0.02*pi:pi;
>> plot(x,cos(x),'k-.',x,sin(x),'b--')
>> axis([-pi,pi,-1,1])
>> xlabel('x=-\pi to \pi','FontSize',16)
>> ylabel('My y  axis','FontSize',16)
>> text(-pi/2,cos(pi/2),'\leftarrowcos(x)=0')
>> title('例:图形标识命令使用')
>> legend(['余弦曲线'],['正弦曲线'])
```

事实上,坐标、刻度、分格线、坐标框以及坐标轴的标注的设置都可以通过图 6.4 所示的坐标轴属性编辑器设置,详细的设置方法不多述了,用户可以自行在坐标属性编辑器中进行设置练习。

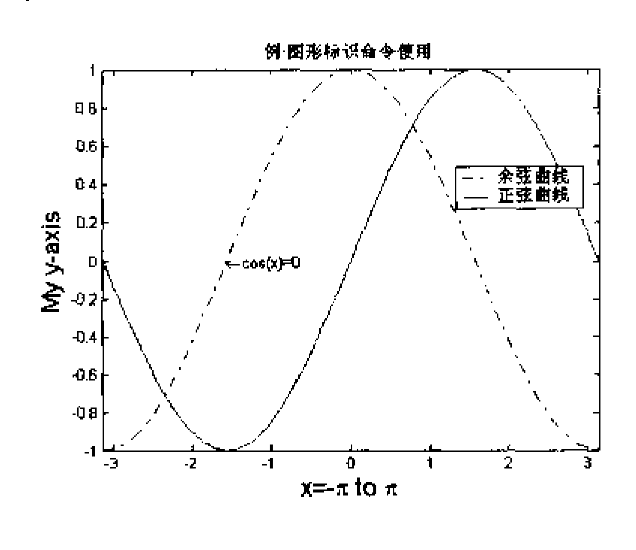


图 6.7 例 6.3 绘制结果

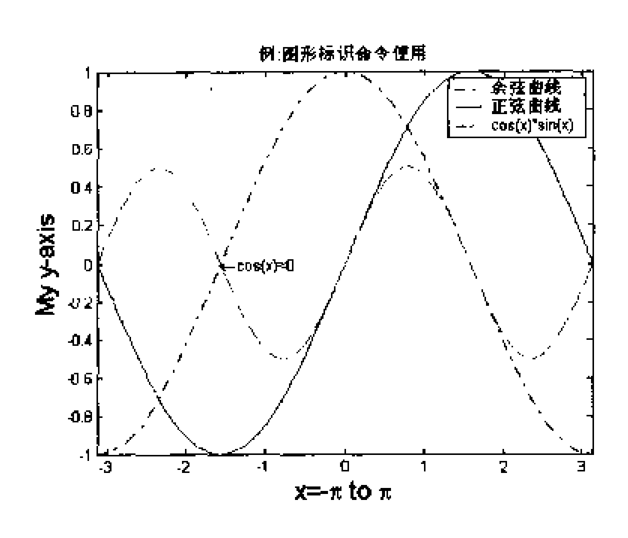


图 6.8 多次叠绘结果

4. 多次叠绘、双纵坐标和多子图

(1) 多次叠绘。例 6.2 和例 6.3 都说明了利用 plot 指令可以在一个图形窗口上同时绘制出多条曲线。在实际工作中,常常会碰到需要在已绘制了曲线的图形窗口中再绘制一些曲线。MATLAB 提供了下列指令:

hold on:保持当前的图形不被刷新,允许在当前图形状态下绘制其他图形,即在同一图形窗口中绘制多幅曲线。

hold off:释放当前的图形,绘制的下一幅图形将作为当前图形,即覆盖原来图形。

hold: 当前图形是否具备刷新功能的双向切换开关。

例 6.4 在例 6.3 的程序最后加入下列几条指令,则可在图 6.7 的基础上加入 y=cos(x).*sin(x),如图 6.8 所示。

```
>> hold on
>> y=cos(x).*sin(x);
>> plot(x,y,'r--')
>> legend(['余弦曲线'],['正弦曲线'],['cos(x)*sin(x)'])
```

(2) 双纵坐标图。MATLAB 提供的 plotyy 指令能够解决实际工作中常常遇到的另外一种需要。plotyy 指令允许用户将同一自变量的两个不同量纲、不同数量级的函数量的变化绘制在同一张图上。

plotyy 的基本指令包括：

plotyy(x1,y1,x2,y2)　以左、右不同纵轴绘制 x1 - y1,x2 - y2 两条曲线；

plotyy(x1,y1,x2,y2,FUN)　以左、右不同纵轴将 x1 - y1,x2 - y2 绘制成 FUN 指定形式的两条曲线；

plotyy(x1,y1,x2,y2,FUN1,FUN2)　以左、右不同纵轴将 x1 - y1,x2 - y2 绘制成 FUN1,FUN2 指定的不同形式的两条曲线。

需要注明的是 FUN,FUN1 和 FUN2 可以是 MATLAB 中所有接受 X - Y 数据对的二维绘图指令。

例 6.5　将系统 $\varphi(s)=\dfrac{10}{s+1}$ 的幅频特性和相频特性绘制在一张图中，如图 6.9 所示。

```
>> x=0.01:0.002:100;
y=20 * log10(10./sqrt(1+x. * x));
z=-atan(x) * 57.3;
plotyy(x,y,x,z,@semilogx)
xlabel('Frequency')
ylabel('Amplitute')
gtext('Amplitute')
gtext('Phase')
ylabel('Phase')
```

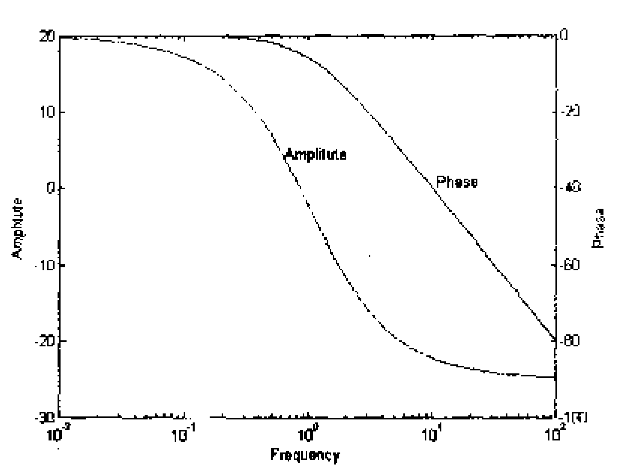


图 6.9　例 6.5 显示结果



(3) 多子图。MATLAB 允许用户在一个图形窗口中布置几幅独立的子图，其基本指令是

subplot(m,n,k)或 subplot(mnk)，表示使 m×n 幅子图中的第 k 幅成为当前图。

例 6.6　subplot 命令使用说明，绘制结果如图 6.10 所示。

```
>> t=0:pi/20:2 * pi;
>> [x,y]=meshgrid(t);
>> subplot(221),plot(sin(t),cos(t))
>> axis([-1 1 -1 1])
>> z=sin(2 * x)+cos(2 * y);
>> subplot(222),plot(t,z)
>> axis([0 2 * pi -2 2])
>> z=sin(x).^3. * cos(y);
>> subplot(223),plot(t,z)
>> axis([0 2 * pi -1 1])
>> z=(sin(x).^3)-(cos(y).^3);
>> subplot(224),plot(t,z)
```



```
>> axis([0 2 * pi  -1 1])
```

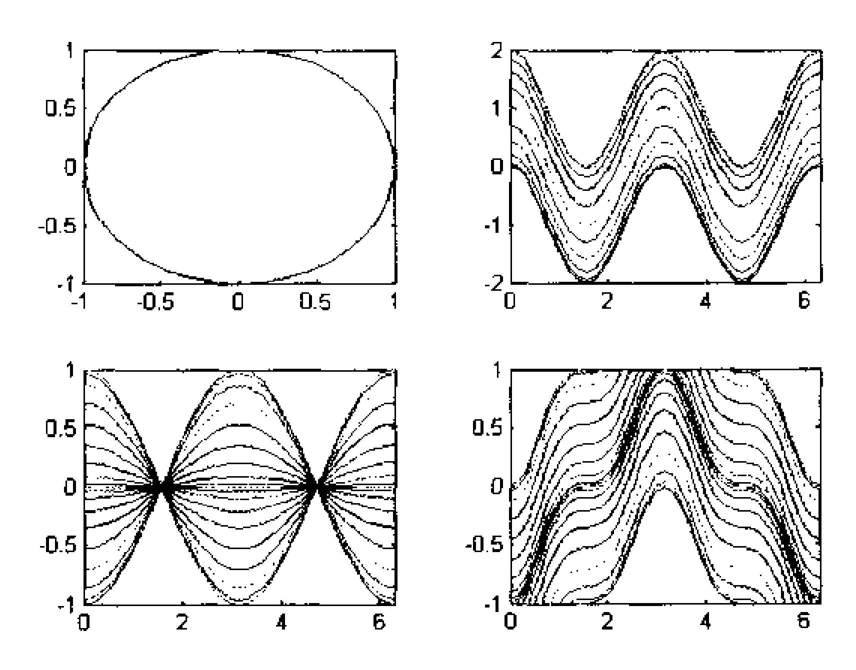


图 6.10　例 6.6subplot 指令绘图结果

6.1.4　特殊二维绘图函数

在 MATLAB 中，用户可以方便地绘制一些特殊的二维图形。可以绘制的特殊二维图形及其命令见表 6.5。

表 6.5　特殊二维绘图函数

指　令	含　　义	指　令	含　　义
bar	直方图	stairs	阶梯图
comet	建立彗星流动图	stem	离散杆图
errorbar	图形加上误差范围	fill	实心图
fplot	较精确的函数图形	feather	羽毛图
polar	极坐标图	compass	罗盘图
hist	累计图	quiver	向量场图
rose	极坐标累计图	pie	饼图

例 6.7　部分特殊二维绘图函数的应用举例，绘制图形如图 6.11 所示。

```
>> t=-10:1:10;
>> subplot(2,2,1);
>> bar(t,cos(t));
>> subplot(2,2,2);
>> compass(t,cos(t));
>> subplot(2,2,3);
```

```
>> rose(t,cos(t));
>> subplot(2,2,4);
>> fill(t,cos(t),'b');
```

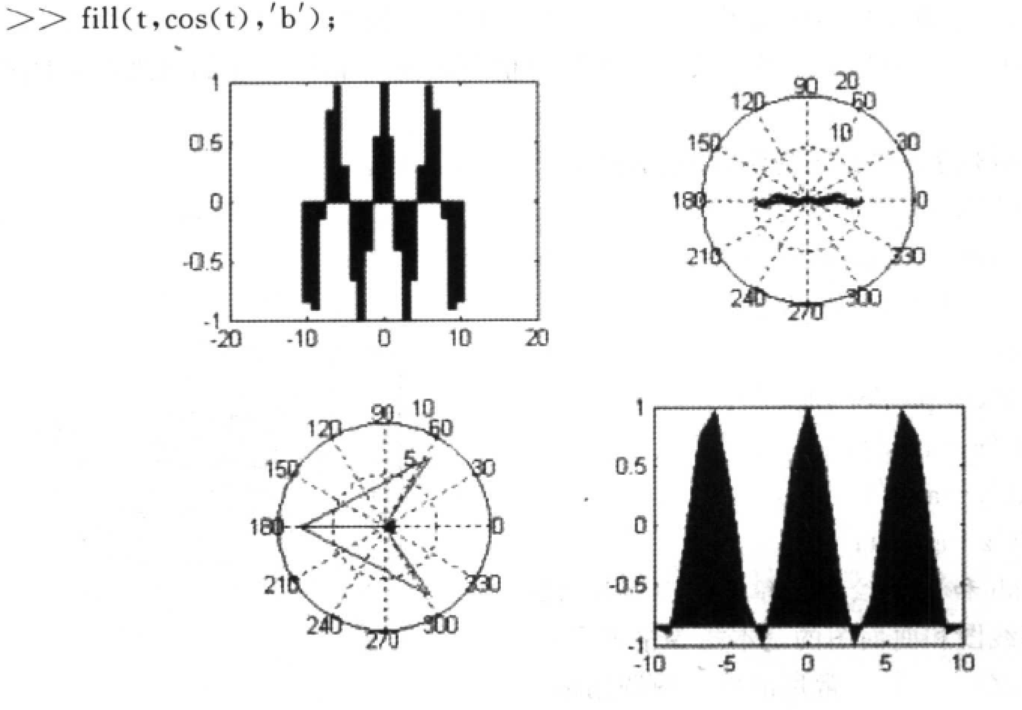


图 6.11　例 6.7 特殊二维绘图函数的绘图结果

6.2　三维图形的绘制

6.2.1　基本绘图函数

三维图形的基本绘图函数见表 6.6。

表 6.6　三维图形的基本绘图函数

命　令	含　义
plot3	将绘制二维图形函数 plot 的特性扩展到三维空间
mesh	三维网线图
surf	三维曲面图

6.2.2　三维图形绘图的基本操作

1. 利用 plot3 指令绘制三维图形

plot3 的基本调用格式：

plot3(x,y,z,'str')输出以向量 x,向量 y 和向量 z 分别为 x,y 和 z 轴的图形，x、y、z 必须

具有相同的长度。

plot3(x1,y1,z1,'str1',x2,y2,z2,'str2',…)在一幅图中，用'str1''指定的方式，输出以 x1、y1、z1 分别为 x、y 和 z 坐标的图形。用'str2'指定的方式，输出以 x2、y2、z2 分别为 x、y 和 z 坐标的图形。其中 str，str1，str2 的意义与二维图形的情况完全相同。它们用来指定线型、色彩和数据点型。

例 6.8　plot3 指令的使用举例，绘制曲线如图 6.12 所示。



```
>> t=0:pi/50:10*pi;
>> plot3(sin(2*t),cos(2*t),t)
>> axis([-1 1 -1 1 0 32]);
>> grid on
>> title('Example for plot3')
>> xlabel('x-axis');
>> ylabel('y-axis');
>> zlabel('z-axis');
```

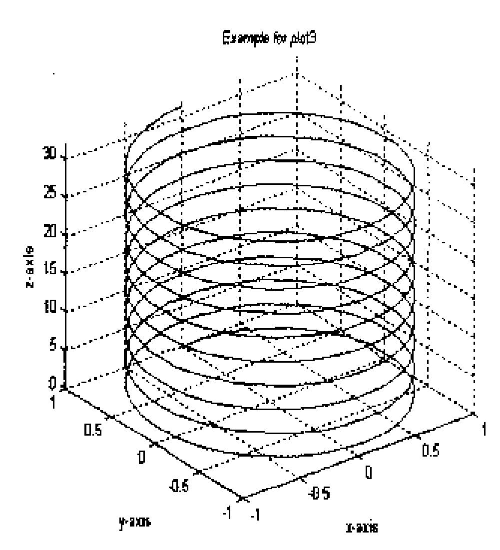


图 6.12　例 6.8plot3 指令的绘图结果

2. 利用 mesh 和 surf 绘制三维网线图和曲面图

(1) 绘制网线图和曲面图的基本指令格式是：

mesh(X,Y,Z)　　　常用的网线图调用格式

mesh(X,Y,Z,C)　　完整的调用格式，画出由 C 指定用色的网线图

surf(X,Y,Z)　　　常用的曲面图调用格式

surf(X,Y,Z,C)　　完整的调用格式，画出由 C 指定用色的曲面图

其中，在完整调用格式中，四个输入量必须是维数相同的矩阵。它们要求 X 和 Y 是自变量“格点”矩阵；Z 是格点上的函数矩阵；C 是指定各点用色的矩阵，可以缺省。缺省时，默认着色矩阵是 Z，即 C=Z。

三维网线图和曲面图的绘制比曲线图稍稍复杂一些。主要是因为它们要求 X 和 Y 是自变量“格点”矩阵，因而在数据准备时，需要生成“格点”矩阵。

(2) 三维图形的数据准备。绘制三维网线图和曲面图以前，需要做如下数据准备：

第一步，确定自变量取值 x=x1:dx:x2;y=y1:dy:y2;

第二步，生成平面上的矩阵 X 和 Y。

方法一，“格点”矩阵的原理性形成法

X=ones(size(y))*x;Y=y'*ones(size(x));

方法二，利用 MATLAB 指令生成法

[X,Y]=meshgrid(x,y);

第三步，计算自变量“格点”上的函数值，即 Z=f(X,Y)。

(3) 三维网线图和曲面图的绘制。

例 6.9　三维网线图和曲面图绘制举例，绘制结果如图 6.13 所示。

```
>> [X Y]=meshgrid([-8:0.5:8]);
>> Z=sqrt(X.^2+Y.^2);
>> subplot(121)
```

```
>> mesh(X,Y,Z)
>> title('三维网线图例')
>> subplot(122)
>> surf(X,Y,Z)
>> title('三维曲面图例')
```

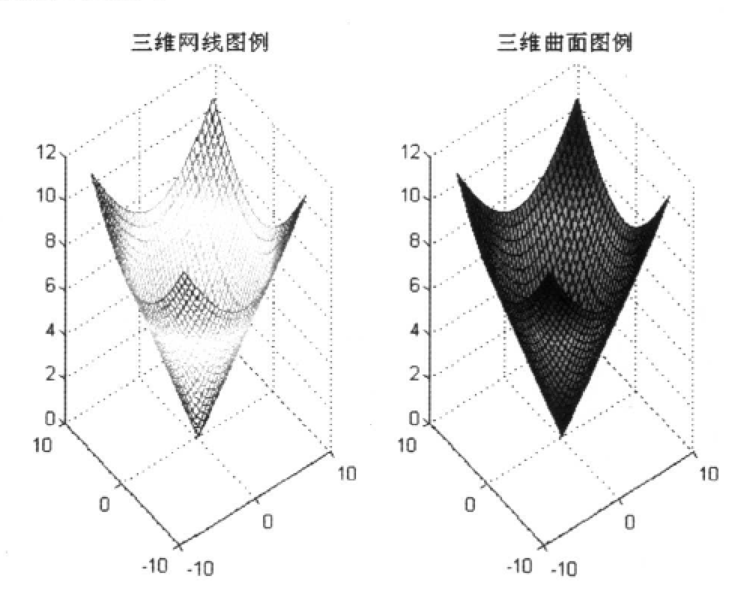


图 6.13　例 6.9 三维网线图和曲面图

3. 特殊三维绘图函数

同二维坐标类型，三维坐标中也有一些特殊的绘图函数。类比二维曲线的绘图指令 plot 和三维曲线的绘图指令 plot3，特殊三维绘图函数的绘制指令很多是在二维特殊绘图函数的基础上加 3。常用的特殊三维绘图函数见表 6.7。

表 6.7　常用特殊三维绘图函数

指　令	含　义	指　令	含　义
bar3	三维直方图	stem3	三维离散杆图
fill3	三维实心图	pie3	三维饼图

习　题

6.1　产生数据 t=0:0.02*pi:2*pi，绘制图形：

(1) 在第一个图形窗口绘制以 t 为自变量的正弦函数；在第二个图形窗口以 t 为自变量的余弦函数；

(2) 在第三个图形窗口用不同的线型绘制以 t 为自变量的正弦函数和余弦函数，并用 legend 命令标注清楚图形的名称。读者可以在此图形中练习 MATLAB 中的各种图形标注函数(如重新设置坐标轴的范围、坐标轴标注等)。

6.2　产生数据 t=0:0.02*pi:pi，y1=sin(2t)，y2=cos(2t)。将图形窗口绘制分成 2 行 2 列的 4 个子图；在前两个子图中分别绘制 t 为自变量 y1，y2 的曲线；在第 3 个子图中绘制 t

为自变量，y=sin(2t) * cos(2t)曲线，在第 4 个子图中绘制 y1 为横坐标，y2 为纵坐标的曲线。并给各个子图加标题。

6.3 使用 MATLAB 三维图形的基本绘图命令，试绘制出

$$z=f(x,y)=\frac{1}{\sqrt{(1-x)^2+y^2}}+\frac{1}{\sqrt{(1+x)^2+y^2}}$$

的三维图。

第七章　M 文件与 M 函数

MATLAB 语言作为一种功能强大的、高级高效的编程工具软件，使用和调试起来都比较容易。利用 MATLAB 提供的丰富的数学函数，用户可以迅速展开工作，编程验证自己的设计和算法。M 文件是 MATLAB 所特有的使用该语言编写的磁盘文件。所有的 M 文件都是以“.m”作为扩展名的。本章介绍编写和调试 MATLAB 语言的工具：M 文件编辑器，MATLAB 的基本语言结构，MATLAB 语言的两种组织形式：MATLAB 脚本和 MATLAB 函数。

7.1　M 文件编辑器

点击 MATLAB 指令窗口工具条上的 New File 图标 或在 MATLAB 命令窗口键入 edit，就可打开如图所示的 MATLAB 文件编辑器 MATLAB Editor，如图 7.1 所示。用此编辑器可以建立、编辑修改和调试 M 文件。

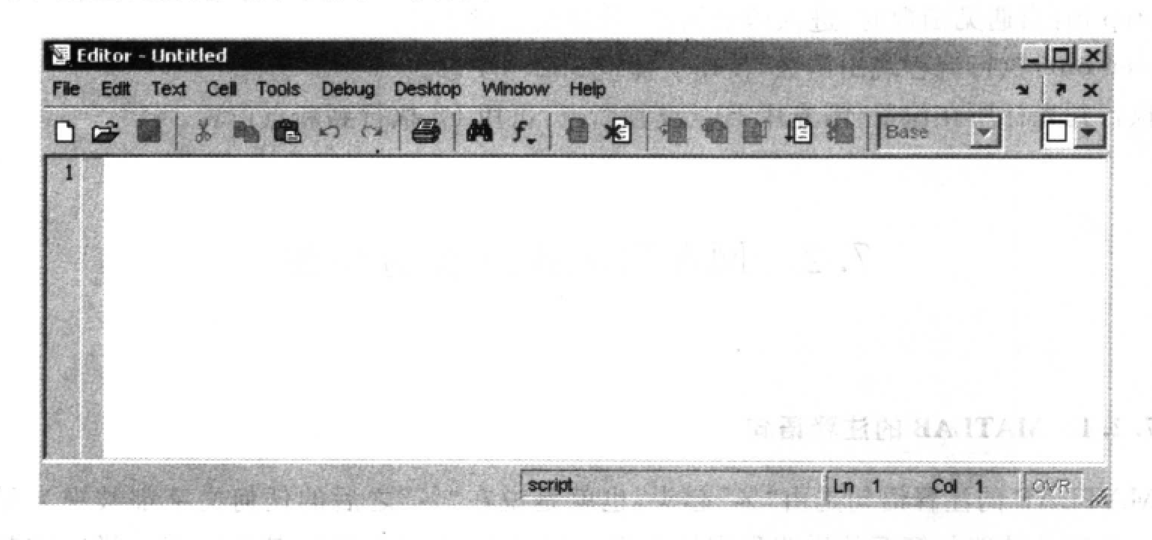


图 7.1　MATLAB 文件编辑器

用户只需在该编辑器中输入程序代码后，在 File 菜单下选择 Save 命令，就会出现保存文件的对话框，指定文件名后即可保存输入的内容，这样就建立了一个新的 M 文件。

7.1.1　MATLAB Editor 的编辑功能

MATLAB Editor 编辑功能的实现主要由 Edit 和 Text 菜单下的各种命令完成的。其使

用方式与 Word 及其他语言的编辑器相似，在此就不赘述。下面介绍一些热键，以帮助读者更快地完成编辑工作。

拷贝粘贴：常用命令 Ctrl+C 和 Ctrl+V。

寻找替代：寻找字符串时点击或使用 Ctrl+F 键显然比用鼠标点击菜单方便。

查看函数：阅读大的程序常常需查看有哪些函数并跳到感兴趣的函数位置，M 文件编辑器提供了一个简单的函数查找快捷按钮：工具条上的图标(Show function)，单击该按钮，会列出该 M 文件所有的函数。

注释：Ctrl+R 和 Ctrl+T 可以多行同时注释，只要先选中须注释的文字，Ctrl+R 即可全部注释，Ctrl+T 可以取消注释。

缩进：缩进格式可以为用户提供清晰的程序结构，编程时应该使用不同的缩进量。增加缩进量用 Ctrl+]，减少缩进量用 Ctrl+[。当一段程序比较乱时，使用 Text 菜单下的 smart indent 或 Ctrl+I 可以快速缩进。

7.1.2 MATLAB Editor 的调试功能

M 文件调试功能的热键与其他编程语言(特别是 VC)的调试热键类似。如果在 Breakpoints 菜单下设置了 stop if error，则程序的执行会停在出错的位置，并在 MATLAB 命令窗口显示出错信息。常用的调试方法如下：

设置或清除断点：使用快捷键 F12 或工具条上的图标。

执行：快捷键 F5 或工具条上的图标。

单步执行：快捷键 F10。

step in：当遇见函数时，进入函数内部，使用快捷键 F11。

step out：执行流程跳出函数，使用快捷键 shift+F11。

执行到光标所在位置：需先用 F12 设置断点，再用 F5 执行到断点位置。

7.2 MATLAB 语言的语法

7.2.1 MATLAB 的注释语句

MATLAB 的注释语句是由“%”起头，也就是说在“%”之后的任何文字都被视为程序的注解。注解的功能是简要的说明程序的内容。编写 M 文件和编写其他程序一样应该养成良好的程序注释习惯，适量的注解可以帮助用户更快、更准确地了解程序。

7.2.2 中断语句

格式：

```
break
```

终止一个循环语句的执行过程。常常利用 break 命令跳出 for，while 循环。

7.2.3　暂停语句

格式：

pause

pause(n)

其中，pause是程序暂时停止运行，直到按下回车键，继续执行程序。而pause(n)是中断n秒后，程序自动继续执行。另外，MATLAB采用热键Ctrl+C中止执行中的MATLAB程序。

7.2.4　MATLAB流程控制语句

MATLAB提供了简明的流程控制语句以便用户使用。流程控制语句的关键词见表7.1。

表7.1　MATLAB流程控制语句关键词

关键词	含义
for	指定次数的循环
while	不指定次数的循环
break	终止循环
if	条件执行语句
elseif	if条件语句
else	if条件语句
end	终止作用域
switch	开关语句
case	列出不同的情况
otherwise	否则语句
return	返回到调用函数

下面将对流程控制语句分别介绍。

1. *循环控制*

许多问题中都需要用到循环控制。例如：迭代求根、循环计算等。几乎所有实用的程序都包含循环。MATLAB中实现循环的语句有两种：for循环语句和while循环语句。

(1) for循环语句。for循环语句允许一组命令以固定的和预定的次数重复，一般用于循环次数已经确定的情况。

for循环语句语法格式为：

```
for  变量=表达式
          语句集合
end
```

其中，变量是循环变量。在表达式中规定该循环变量的初始值、步长和终值。如k=1:2:50。步长可以为负。如果步长为1，则可缺省。对于正步长，当变量的值大于终值时，结束循环；对于负步长，当变量的值小于终值时，结束循环。

for 循环可按需要嵌套使用，但需注意每一个“for”关键词必须和一个“end”关键词配对，否则会出错。其语法如下：

```
for  变量 1=表达式 1
……
  for  变量 2=表达式 2
     语句集合
  end
……
end
```

例 7.1　for 循环嵌套的实例：在 M 文件编辑器中输入下列程序，计算 1～6 的乘法表。

解

```
for i=1:6
  for j=1:i
     p(i,j)=i*j;
  end
end
```

查看计算结果：在 MATLAB 命令窗口键入变量名 p，得

```
>> p
p=
     1     0     0     0     0     0
     2     4     0     0     0     0
     3     6     9     0     0     0
     4     8    12    16     0     0
     5    10    15    20    25     0
     6    12    18    24    30    36
```

(2) while 循环语句。while 循环语句一般用于事先不能确定循环次数的情况。

while 循环语句，流程图如图 7.2 所示，其语法格式如下：

```
while 逻辑表达式
     语句集合
end
```

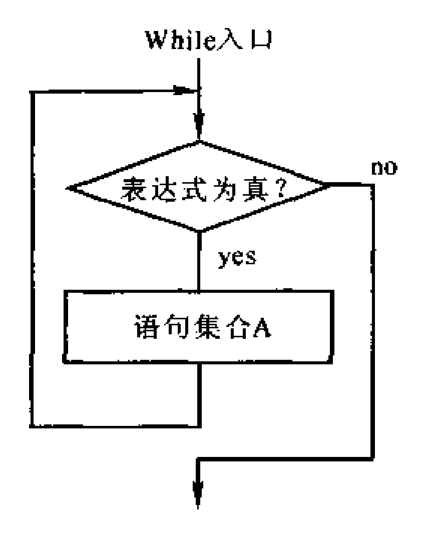


图 7.2　while 语句流程图

当表达式的值为真时，执行语句集合；当表达式的值为假时，结束该循环。当表达式的计算对象为矩阵时，只有当矩阵中所有的元素均为正时，才执行语句集合。当表达式为空矩阵时，不执行语句集合中的任何语句。为明了起见，可用函数 all 和 any 等把矩阵表达式转换成标量。

需要再次指出的是，用户可以利用 break 命令跳出 for 和 while 循环。

2. 条件控制

MATLAB 中有两种条件控制语句：if - else - end 条件执行语句和 switch - case 多分支语句。

(1) if - else - end 分支语句。MATLAB 中由 if 语句进行判断，其调用格式有 3 种。

格式一　基本调用格式如下：

```
if 逻辑表达式
   语句集合
end
```

流程图如图7.3(a)所示。

if和逻辑表达式之间应留空格。if语句将计算逻辑表达式的值，若逻辑表达式的值为真(非零)，则执行语句集合，执行完之后继续向下执行；若逻辑表达式的值为假，则直接向下执行。若逻辑表达式是空矩阵时，MATLAB认为条件为假。

格式二　如果有两种选择，需采用如下的调用格式：

```
if 逻辑表达式
   语句集合1
else
   语句集合2
end
```

其流程图如图7.3(b)所示。

若逻辑表达式的值为真(非零)，则执行语句集合1，然后跳出条件块；若逻辑表达式的值为假，则执行语句集合2，之后也跳出条件块。

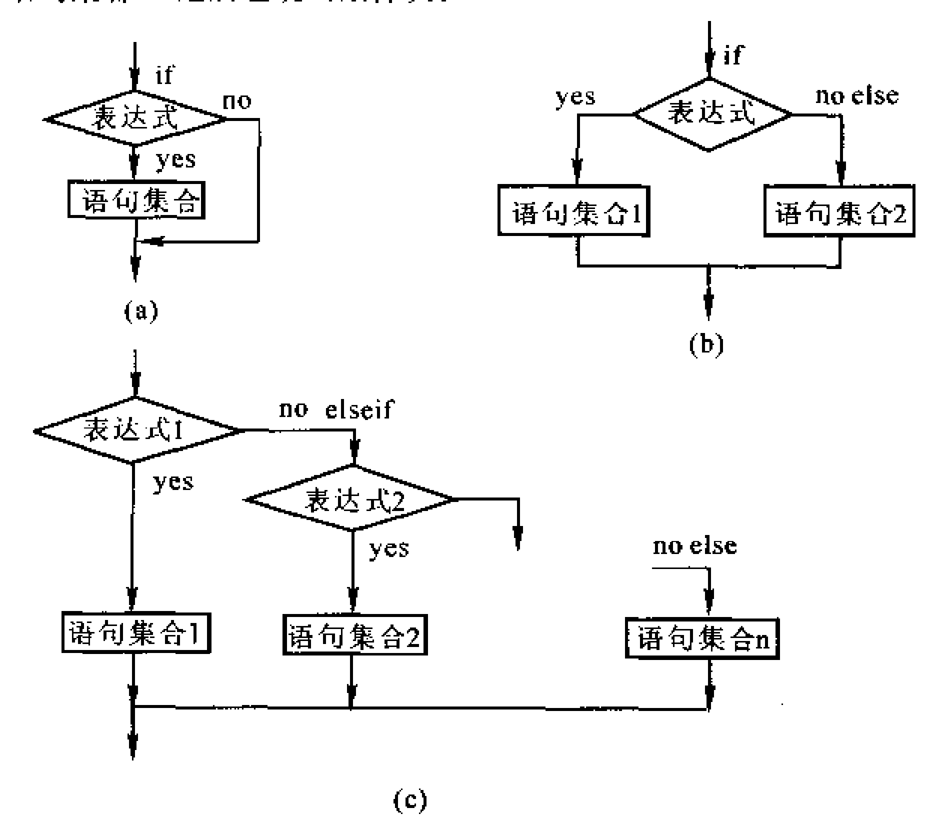


图7.3　if条件语句的流程图

格式三　如果选择项多于两个，采用如下的调用格式：

```
if 逻辑表达式1
   语句集合1
```

```
elseif  逻辑表达式 2
        语句集合 2
……
else
   语句集合 n
end
```

流程图如图 7.3(c)所示。

若逻辑表达式 1 的值为真(非零),则执行语句集合 1,然后跳出条件块;若逻辑表达式 1 的值为假,逻辑表达式 2 的值为真(非零),则执行语句集合 2,之后跳出条件块……若前面所有逻辑表达式的值均为假,则执行语句集合 n(else 后的语句集合)。

例 7.2 求满足 $\sum_{i=1}^{m} i > 10\,000$ 的最小值 。

解 用户可在 M 文件编辑器中键入并运行如下程序:

```
mysum = 0;
for m=1: 1000
   if (mysum>10000), break; end
   mysum = mysum+m;
end
```

运行结果:

```
>> mysum
mysum =
        10011
```

(2) switch - case 语句。

switch - case 语句是多分支语句,其一般调用格式为

```
switch    表达式(标量或字符串)
          case    值 1
                  语句集合 1
          case    值 2
                  语句集合 2
                  ……
          otherwise
                 语句集合 n
end
```

程序运行至 switch 语句时,先计算表达式,当表达式的值(或字符串)与某 case 语句中的值(或字符串)相同时,它就执行该 case 语句后的语句集合,而后直接跳到终点的 end。case 语句可以有任意多个分支,如果没有任何一个 case 值能与表达式的值相符,则将执行 otherwise 后面的语句集合 n。

例 7.3 switch - case 语句应用举例。

解 MATLAB 编辑器中输入如下程序:

```
n=input('please input n:')
switch n
    case 1
        aaa=20;
    case 2
        aaa=40;
    case 3
        aaa=80;
    otherwise
        aaa=90;
end
V=aaa*6;
aaa
V
```

如果输入的 n 值为 2，则 MATLAB 命令窗口显示结果为

```
please input n:2          %用户需输入数据 2。
n =
      2
aaa =
    40
V =
   240
```

7.2.5　MATLAB 的输入与输出语句

1. 输入语句 input

input 指令把 MATLAB 的"控制权"暂时交给用户。此后，用户通过键盘键入数值、字符串或表达式，并经"回车"将键入的内容输入工作空间，同时将"控制权"交还给 MATLAB。常用格式有下列几种。

(1) 输入数据格式：x=input('message')，例如：

指令：x=input('please input a number:')

MATLAB 命令窗口显示：please input a number:22　　　%用户输入数据 22。

x = 22

数据可以是 MATLAB 的各种数据类型：如数值、字符串、元胞数组等。

(2) 输入字符串格式：x=input('message','s')，例如：

指令：x=input('please input a string:','s')

MATLAB 命令窗口显示：please input a string:this is a string

x = this is a string

2. 输出语句

(1) 输出显示命令。

1)自由格式 (disp) 。例如:

指令:disp(23+454--29 * 4)

MATLAB 命令窗口显示:361

指令:disp([11 22 33; 44 55 66; 77 88 99])

MATLAB 命令窗口显示:

11 22 33

44 55 66

77 88 99

指令:disp('this is a string')

MATLAB 命令窗口显示:

this is a string

2)格式化输出 (fprintf)。例如:

指令:fprintf('The area is %8.5f\n', area)　　%此命令要求输出 8 位数含 5 位小数的输出值。

MATLAB 命令窗口显示结果:The area is 12.56637

需要指出的是,输出格式前须有"%"符号,跳行符号须有"\"符号。

(2) 错误消息显示命令:error。例如:

指令:error ('this is an error')

MATLAB 命令窗口显示结果:??? this is an error

7.2.6 文件操作

1. 变量的保存与调用

(1) 变量保存(save)。使用 save 可以将 MATLAB 工作空间的变量保存到文件中,以便以后可以调用这些变量。将 MATLAB 工作空间的变量保存到文件中的指令调用格式为

save filename variables

(2) 变量调用(load)。从文件中调用变量的指令调用格式为

load filename variables

需要说明的是,这两条指令所用文件的扩展名均为 . mat,指令中的 variables 可以缺省,缺省时表示将 MATLAB 工作间的所有变量保存到文件中或调用文件中存储的所有变量。

例 7.4 假设 MATLAB 工作空间中已有两个变量 x 和 y,在命令窗口键入 whos 可以查看,利用 save 命令可以存储变量。

```
>> whos
  Name      Size                 Bytes  Class
  x         1x150                1200   double array
  y         1x1                  8      double array
Grand total is 151 elements using 1208 bytes
>> save examp75  x y      %将 x 和 y 存于当前目录下的 examp75. mat 文件中
```

例 7.5 假设 MATLAB 当前目录下有一个数据文件 examp75. mat,利用 load 命令调用该文件存储的变量。

```
>> load examp75
>> whos      %在MATLAB命令窗口键入whos可调用变量。
  Name       Size          Bytes    Class
  x          1x150         1200     double array
  y          1x1           8        double array
Grand total is 151 elements using 1208 bytes
```

2. 文件的打开与关闭

使用fopen和fclose可以对普通的文件进行打开、关闭及处理。

使用格式：

```
fid=fopen(filename,ftype)
st=fclose(fid)
```

例如，在MATLAB命令窗口键入

```
>>fid=fopen('examp75.mat','r')
fid=
    3
>>st=fclose(fid)
st =
    0
```

3. 文件的输入与输出

(1)不定格式读取。指令调用格式：a=fread(fid,size)。

从文件fid中读取数据保存到矩阵a中，size是一个限制从文件中读取数据的个数的选项，如果缺省，则表示读取文件中的所有数据，如果规定，则size可取值为

N　　表示读取N个数据形成一个列向量；

inf　　表示读取所有的数据；

[M,N]　　表示读取M×N个数据，按列的顺序形成M×N矩阵。N可以是inf，但M不可以是inf。

例如，在MATLAB命令窗口键入

```
>> a=fread(fid)
```

(2) 指定格式读取。指令调用格式：a=fscanf(fid,format,size)。

从句柄fid所指定的文件中，按字符串format所指定的数据格式读取数据，把他们保存到矩阵*a*中。

```
>>str=fscanf(fid,'%s')
str =
MATLAB5.0MAT - file,Platform:PCWIN,Createdon:……
```

(3) 将数据写入文件中。fprintf(fid,format, *A*,*B*,…)。

例如，将字符串所指定的数据写入到由fid所指定的文件中。

```
>> t=0:0.001:1;
>> fid=fopen('ok.mat','w');
>> fprintf(fid,'%d',t);
```

```
>> length(t)
ans =
        1001
>> clear
>> fid=fopen('ok.mat','r');
>> fread(fid);
>> whos
  Name        Size                    Bytes  Class
  ans     12989x1                    103912  double array
  fp          1x1                         8  double array
Grand total is 12990 elements using 103920 bytes
```

再如：下列程序将 ccc 存入文件 csz.dat 中。

```
fid = fopen('csz.dat','w');
fprintf(fid,'%9.10f\n',ccc);
fclose(fid);
```

7.3　脚本文件和 M 函数的编写与调用

7.3.1　脚本文件

脚本文件是命令的集合，是由一系列 MATLAB 命令、内置函数及 M 文件等组成的文件。脚本文件在 MATLAB 编译器中建立，并被保存为 .m 文件，按顺序执行，执行过程中生成的变量存放在当前工作空间中。

例 7.6　编写 M 文件绘制函数 $y(x)=\begin{cases}2|\sin x| & x\leqslant 0\\ x & 0<x\leqslant 3\\ -x+6 & x>3\end{cases}$，在区间[−6　6]中的图形。

解　在 MATLAB 编译器中键入如下命令，保存为 examp7_6.m，点击编辑器工具栏的运行键进行计算并绘图，绘图结果如图 7.4 所示。

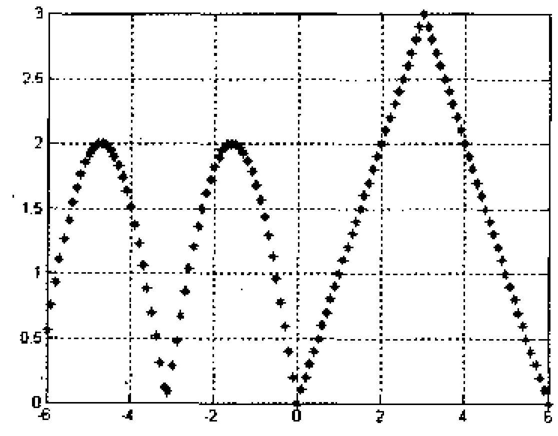


图 7.4　例 7.6 计算绘图结果

```
x=-6:0.1:6
leng=length(x);        %计算 x 向量长度。
for m=1:leng
    if x(m)<=0
        y(m)=2*abs(sin(x(m)));
    elseif x(m)<=3
        y(m)=x(m);
```

```
    else
        y(m)=-x(m)+6;
    end
end
figure
plot(x,y,'*'),grid on
```

7.3.2　M 函数

除了内置函数和工具箱函数外，不同的用户可能需要开发自己专用的函数或通用函数。M 函数(function file)也称子程序，它必须由 MATLAB 调用并具有一定的通用性。

M 函数的格式：function [返回变量表]=函数名(输入变量列表)

　　　　　　　　　　　注释说明语句段

　　　　　　　　　　　函数体语句

调用格式：输出变量=函数名(输入变量)

必须强调的一点是 M 函数文件名必须同函数名。当一个 M 文件中含多个 M 函数时，第一个函数是主函数，M 文件名必须是主函数名。

M 函数调用时，函数输入和输出可以与函数定义的变量不相同。

M 函数注释由"%"开始的行表示，help function_name 显示的是第一行后的注释。

例 7.7　编写通用的函数，计算例 7.6 中函数在任意点的值，并绘制函数在区间[-6,6]中的图形。

解　编写 M 函数 cal 如下所示，并存于同名 M 文件 cal.m 中。

```
function y=cal (x)                %M 文件定义。
  leng=length(x);                 %计算 x 向量长度。
    for m=1:leng
        if x(m)<=0
          y(m)=2*abs(sin(x(m)));
        elseif x(m)<=3
            y(m)=x(m);
        else
            y(m)=-x(m)+6;
        end
      end
```

编写 M 脚本文件 examp7_7，设置 x 取值范围，调用 M 函数 cal，并绘制图形。

```
x=-6:0.1:6;
y=cal(x);
figure
plot(x,y,'*'),grid on
```

习 题

7.1 使用循环语句形成一个元素满足 Fibonacci 规则的数组，即令数组的第 $k+2$ 个元素满足 $a_{k+2}=a_k+a_{k+1}$，$k=1, 2, \cdots$ 且 $a_1=1$，$a_2=1$。现要求出该数组中的第一个大于 10 000 的元素。

7.2 编写 M 函数生成 Hilbert 矩阵，该矩阵是一个 $n\times m$ 阶矩阵，它的第 i 行第 j 列元素值为 $1/(i+j-1)$。编写的函数中需返回此 Hilbert 矩阵 $\mathbf{A}$ 和它的平方 $\mathbf{A}^{\mathrm{T}}\mathbf{A}$。并求出 4×3Hilbert 矩阵及其平方。

7.3 Newton 迭代法是求解非线性方程的常用方法。如果用户想求出 $f(x)$的根，且求出 $f'(x)$，这时若用户首先给出初值 x_0，则方程的根可以由下列迭代公式求出：$x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$，若 $f(x) = a_1x^n + a_2x^{n-1} + \cdots + a_nx + a_{n+1}$，试用 MATLAB 条件语句编写程序求 $f(x) = 0$ 的根，计算 $f(x) = x^5 + 2x^4 + 3x^3 + 4x^2 + 5x + 6$ 的根，并用 roots( ) 命令验证所得结果。(用 eps 内部变量判断迭代结束条件)。

7.4 绘制函数 $f(x) = \begin{cases} \sin x & x > 5 \\ 3 + x & 0 < x \leqslant 5 \\ x^2 & x \leqslant 0 \end{cases}$，在区间[－10，10]的图形。

7.5 编写绘制下列分段函数所表示曲面的 M 函数。

$$p(x_1,\ x_2) = \begin{cases} 0.0545\ 7\mathrm{e}^{-0.75x_2^2-3.75x_1^2-1.5x_1} & x_1 + x_2 > 1 \\ 0.757\ 5\mathrm{e}^{-x_2^2-6x_1^2} & -1 < x_1 + x_2 \leqslant 1 \\ 0.545\ 7\mathrm{e}^{-0.75x_2^2-3.75x_1^2+1.5x_1} & x_1 + x_2 \leqslant -1 \end{cases}$$

并绘制出 $x_1 \in [-2, 2]$，$x_2 \in [-2, 2]$ 时的曲面。

第八章　基于 Simulink 的动态系统仿真入门

Simulink 是基于 MATLAB 的图形化仿真设计环境，是 MATLAB 提供的进行动态系统建模、仿真和综合分析的集成软件包。它使用图形化的系统模块对动态系统进行描述，并在此基础上采用 MATLAB 的计算引擎对动态系统在时域内进行求解。MATLAB 计算引擎主要对系统微分方程和差分方程求解。Simulink 和 MATLAB 是高度集成在一起的，因此，它们之间可以进行灵活的交互操作。

Simulink 可以处理的系统包括：线性、非线性系统；离散、连续及混合系统；单任务、多任务离散事件系统等。在 MATLAB 7.x 版本中，可直接在 Simulink 环境中运作的工具箱和模块包很多，已覆盖航空、航天、通信、控制、信号处理、电力系统、机电系统等诸多领域，涉及的内容专业性极强。

8.1　启用 Simulink 并建立系统模型

因为 Simulink 是基于 MATLAB 的图形化仿真环境，因此启动 Simulink 之前必须首先运行 MATLAB，然后才能启用 Simulink 并建立图形化的系统模型。MATLAB 有两种启动 Simulink 的方式：

命令行方式启动：在 MATLAB 命令窗口键入 Simulink 即可；

快捷方式启动：鼠标点击 MATLAB 工具栏图标 。

启动 Simulink 后，屏幕上会出现 Simulink 主窗口，鼠标点击 Simulink 主窗口工具栏上的□或从主菜单 File 中选择 New\model，即可打开系统模型编辑器，如图 8.1 所示。图 8.1 中由上到下依次是 MATLAB 主窗口、Simulink 主窗口（即 Simulink 库浏览器）和系统模型编辑器，图中箭头反映了操作的顺序。

在打开一个新的系统模型文件后，用户便可从 Simulink 模块库中选择合适的系统模块或自定义的模块来建立系统模型。因此，用户需先了解 Simulink 中内置的系统模块。

8.2　Simulink 模型库简介

为了方便用户快速构建所需的动态系统，Simulink 提供了大量的、以图形形式给出的内

置系统模块。使用这些内置模块可以快速方便地设计出特定的动态系统。图 8.2 所示是 Simulink 的模型库浏览器。由该浏览器可以看出 Simulink 内置模型库包含公共模型库和专业模型库。

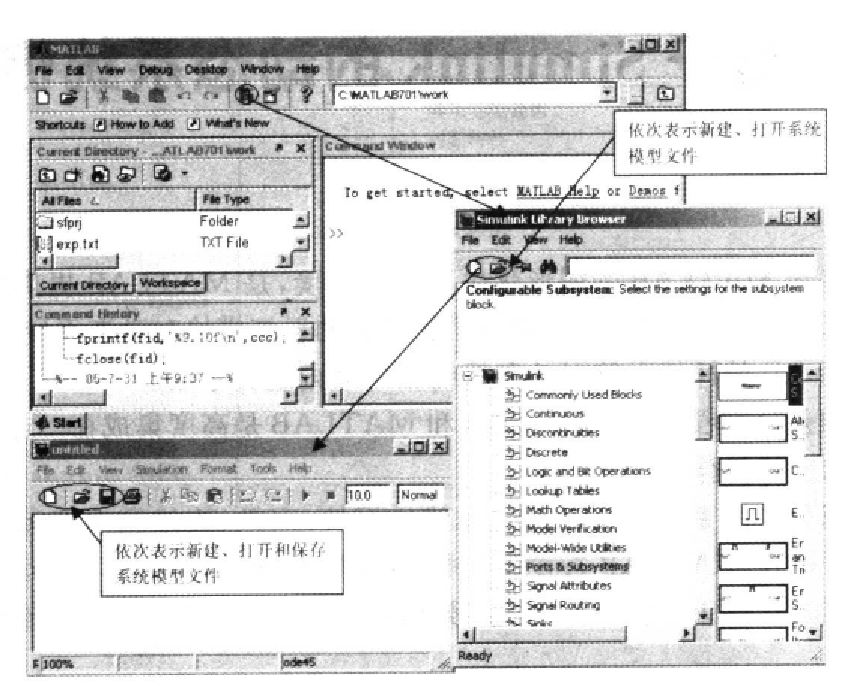


图 8.1　启动 Simulink 并建立系统模型的基本操作

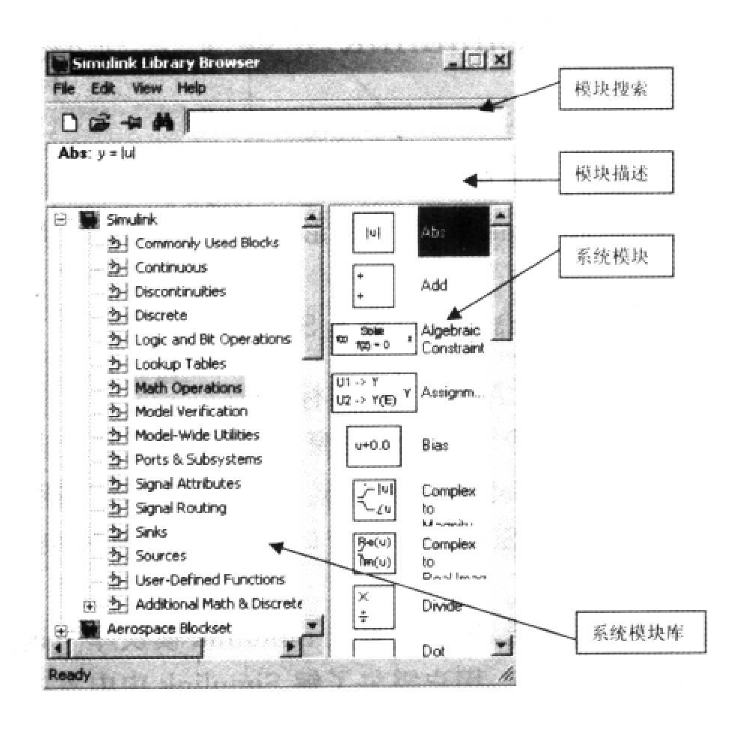


图 8.2　Simulink 模型库浏览器

8.2.1　Simulink 的公共模型库

Simulink 的公共模型库是 Simulink 中最通用的模型库，如图 8.3 所示，它可以应用到不同专业，包括 16 个子模块库。下面对各子模型库及其功能做简要介绍。

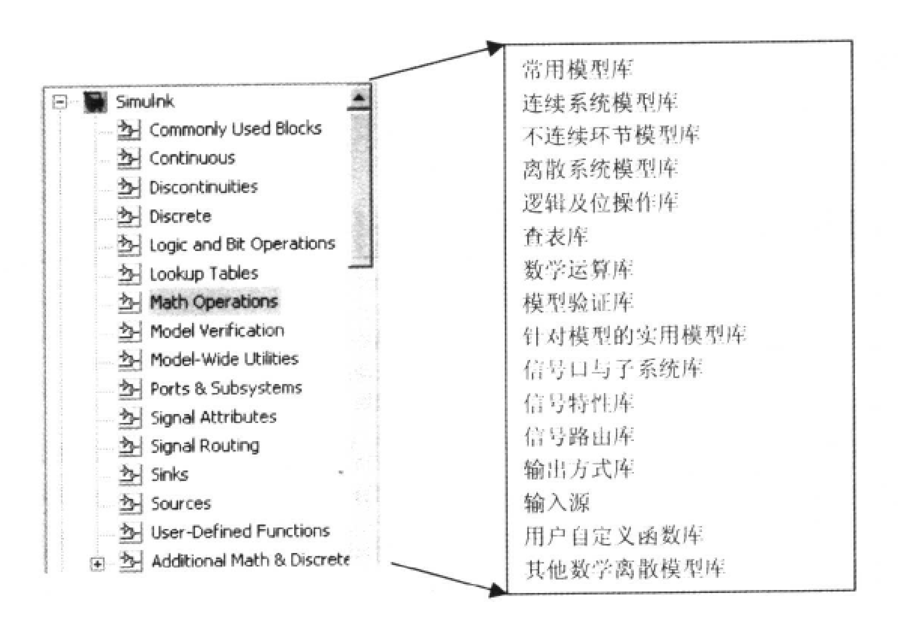


图 8.3　Simulink 的公共模型库

1. Commonly Used Blocks(常用模型库)

在新版本的 Simulink 中，为了方便用户使用，专门将常用的 22 种模块放在了常用模型库中。这些常用的模型包括输入模块、输出显示模块、连续(离散)系统积分模块、数学运算模块及信号路由模块等，其功能将在后面的各种模型库中介绍。

2. Continuous(连续系统模型库)

连续系统模型库及其各模块的功能如图 8.4 所示。

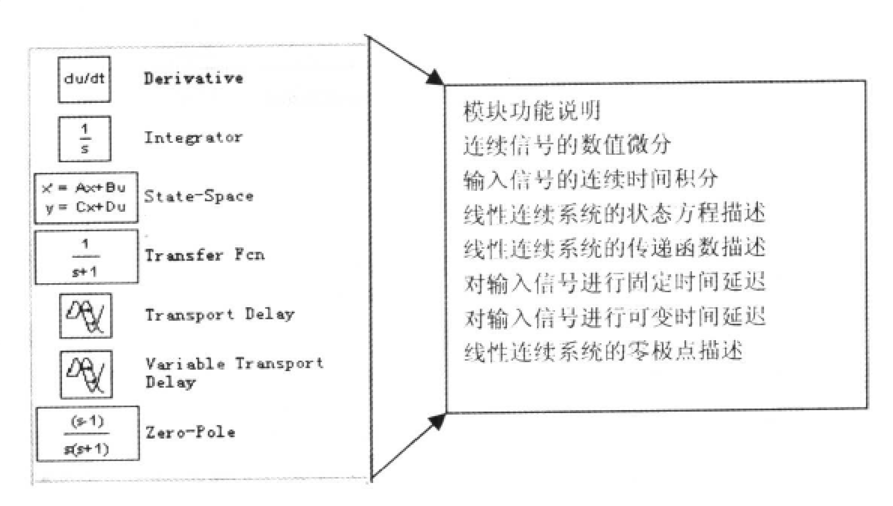


图 8.4　连续系统模型库及其模块功能

3. Discontinuities(不连续环节模型库)

不连续环节模型库及其各模块的功能如图 8.5 所示。

4. Discrete(离散系统模型库)

离散系统模型库及其各模块的功能如图 8.6 所示。

5. Logic and Bit Operations(逻辑及位操作库)

逻辑及位操作库及其各模块的功能如图 8.7 所示。

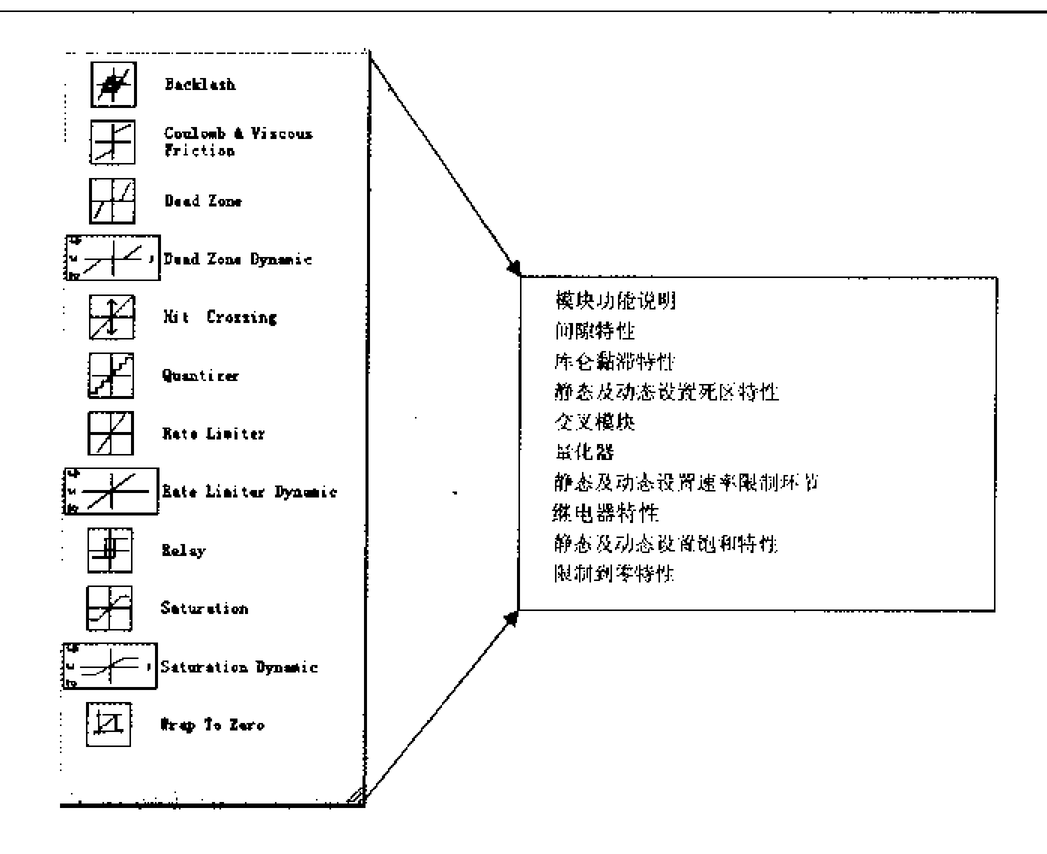


图 8.5 不连续环节模型库及其模块功能

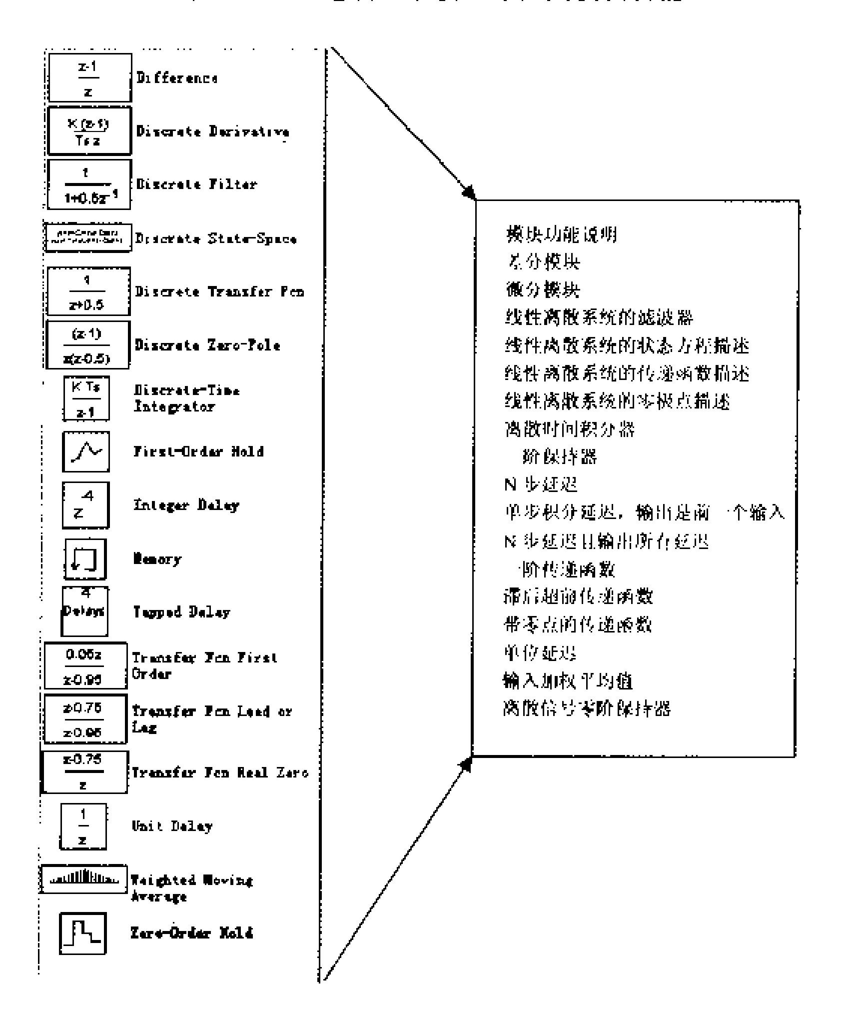


图 8.6 离散系统模型库及其模块功能

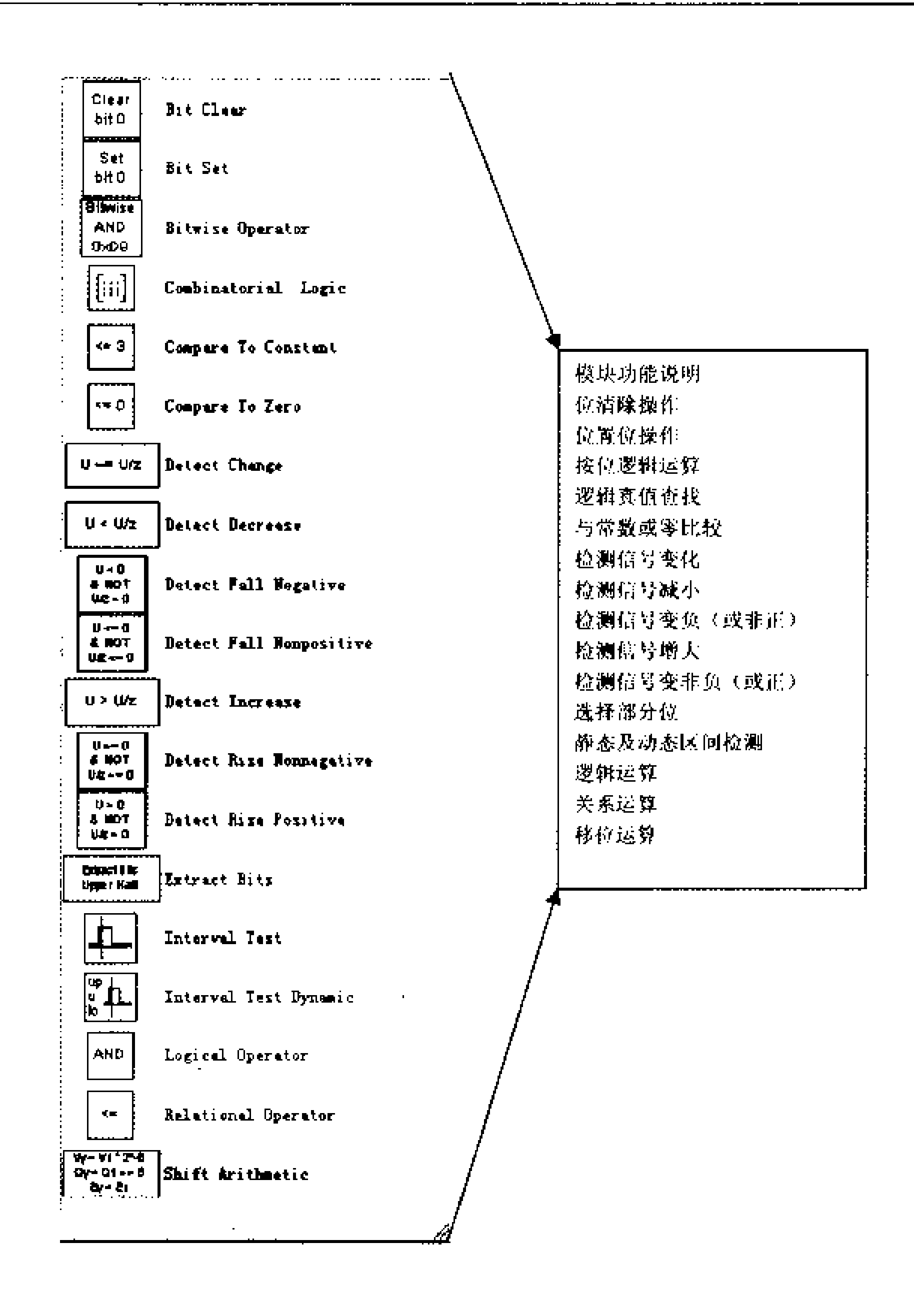


图 8.7　逻辑及位操作库及其模块功能

6. Look up Tables(查表库)

查表库及其各模块的功能如图 8.8 所示。

7. Math Operation(数学运算库)

数学运算库及其各模块的功能如图 8.9 所示。

8. Model Verification(模型验证库)

模型检验及其各模块的功能如图 8.10 所示。

9. Model-wide Utilities(针对模型的实用模型库)

针对模型的有用功能模块及其各模块的功能如图 8.11 所示。

10. Ports and Subsystem(信号口与子系统)

信号口与子系统库及其各模块的功能如图 8.12 所示。

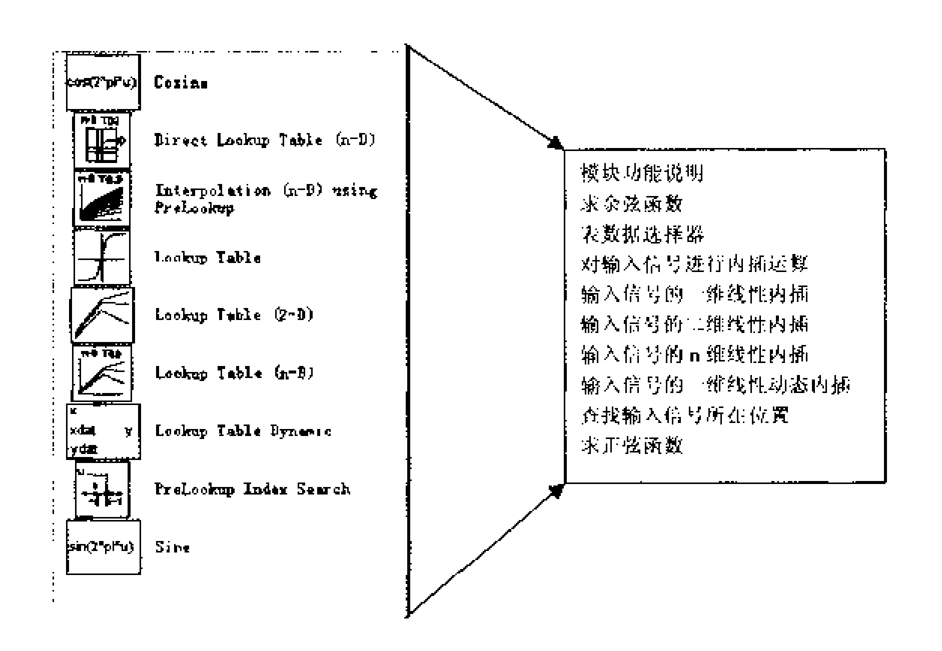


图 8.8 查表库及其模块功能

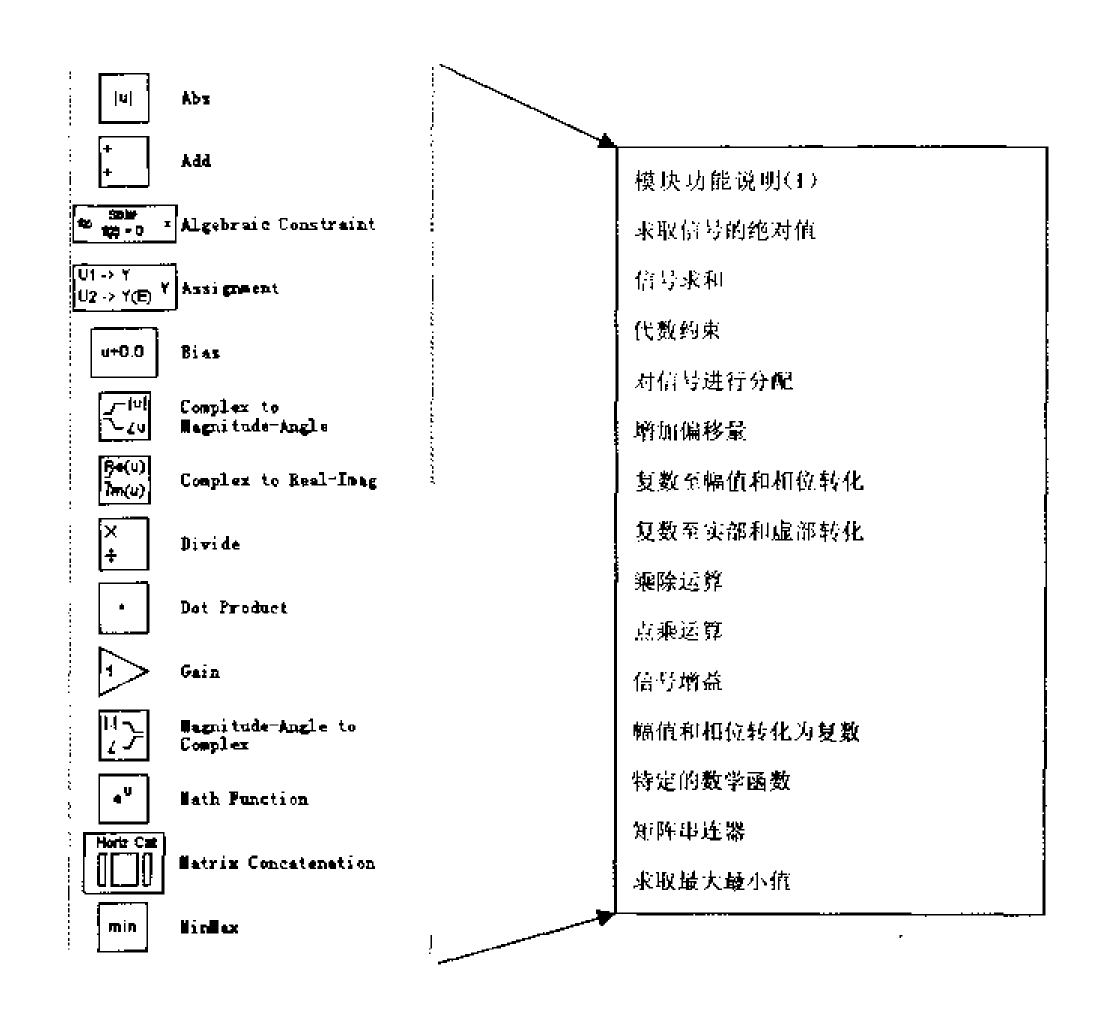


图 8.9 数学运算库及其模块功能

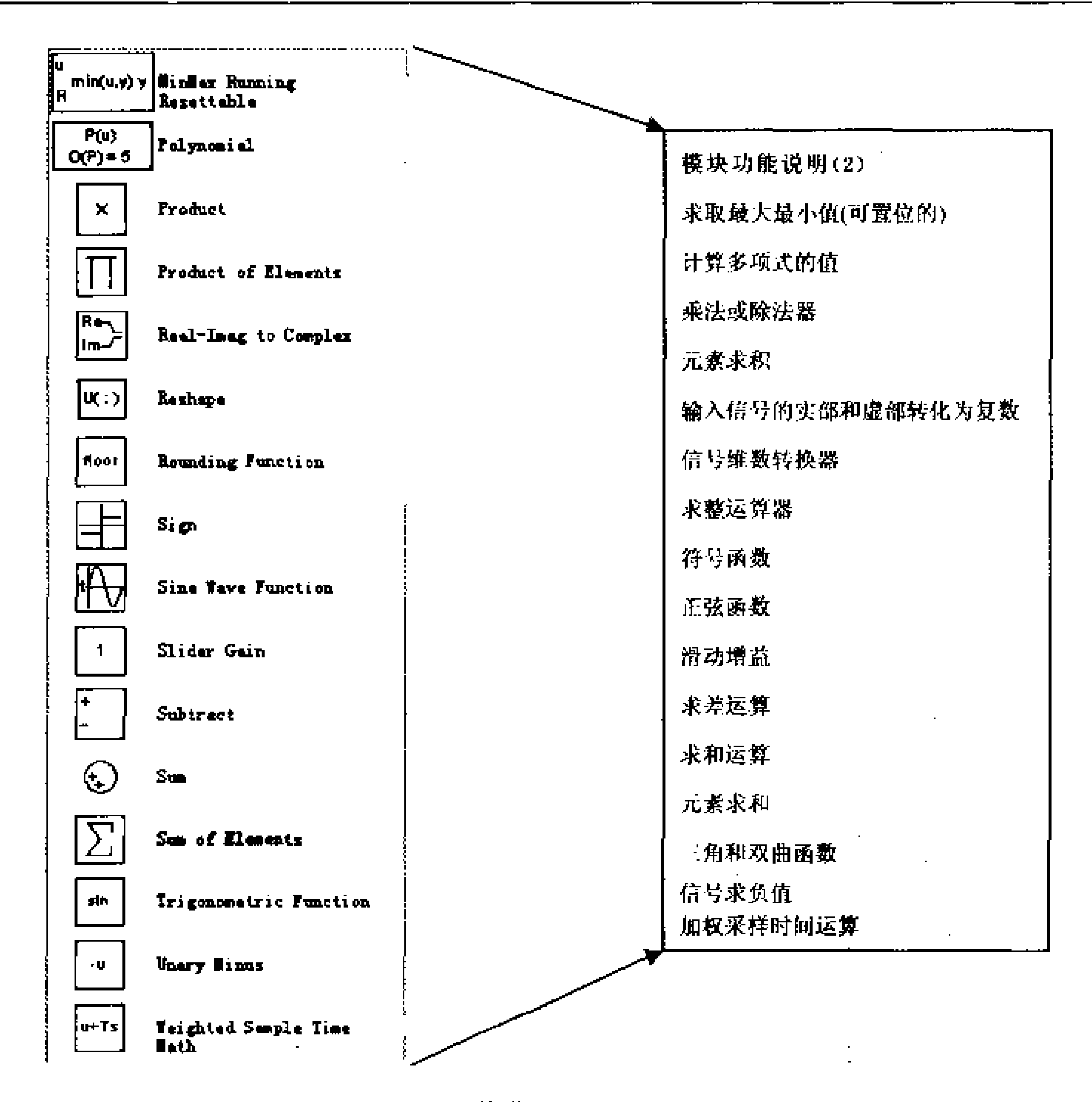


图 8.9(续) 数学运算库及其模块功能

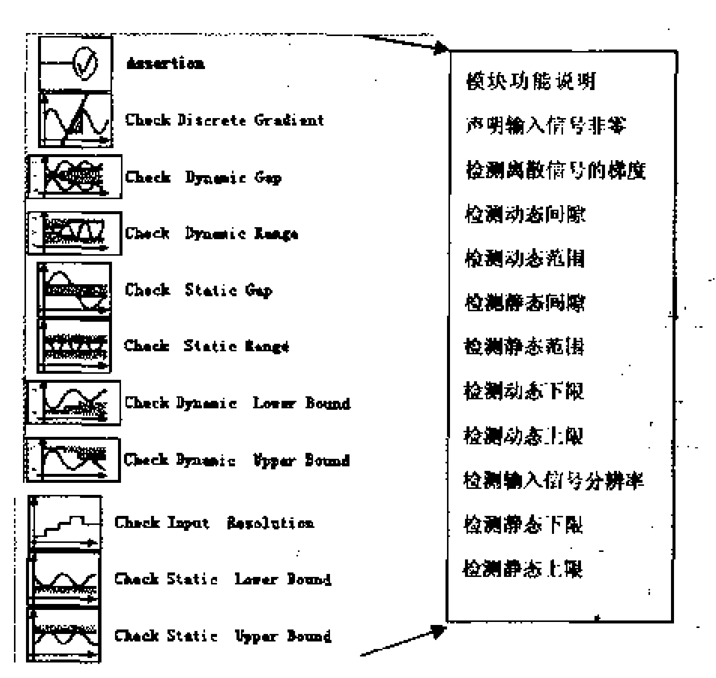


图 8.10 模型检验及其模块功能

11. Signal Attributes(信号特性库)

信号特性库及其中各模块的功能如图 8.13 所示。

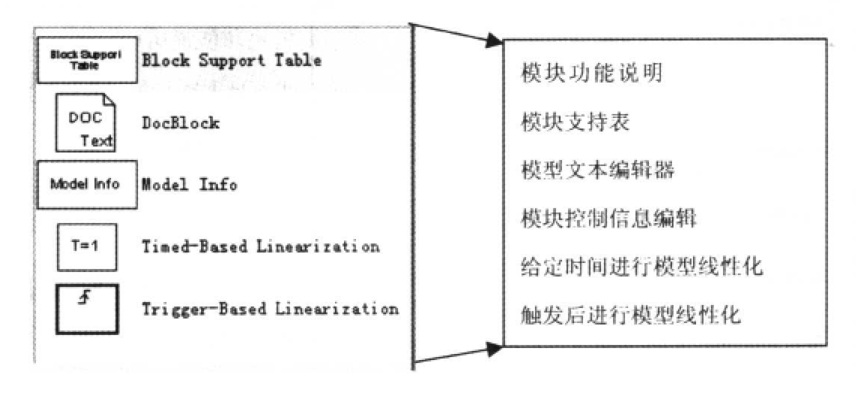


图 8.11　针对模型的实用模型库及其模块功能

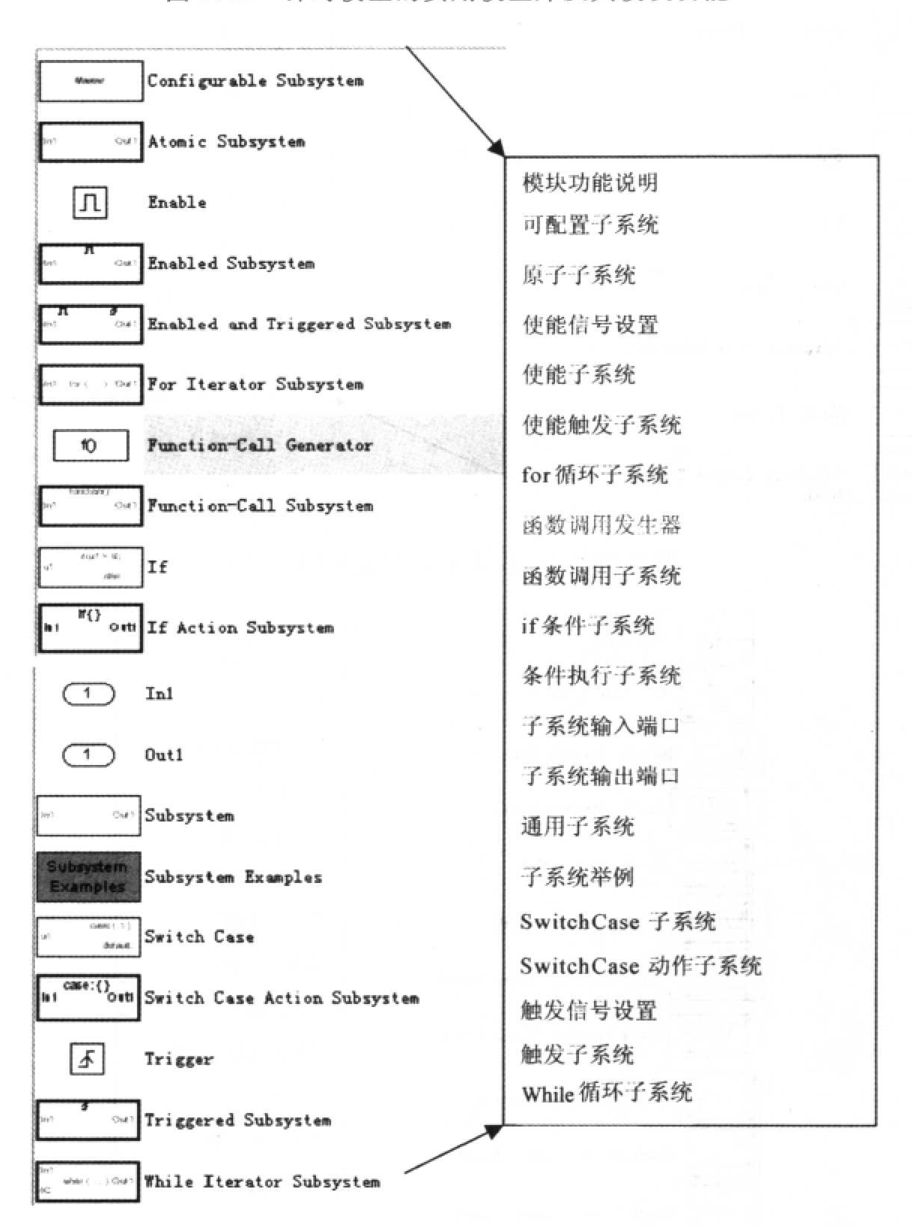


图 8.12　信号口与子系统库及其模块功能

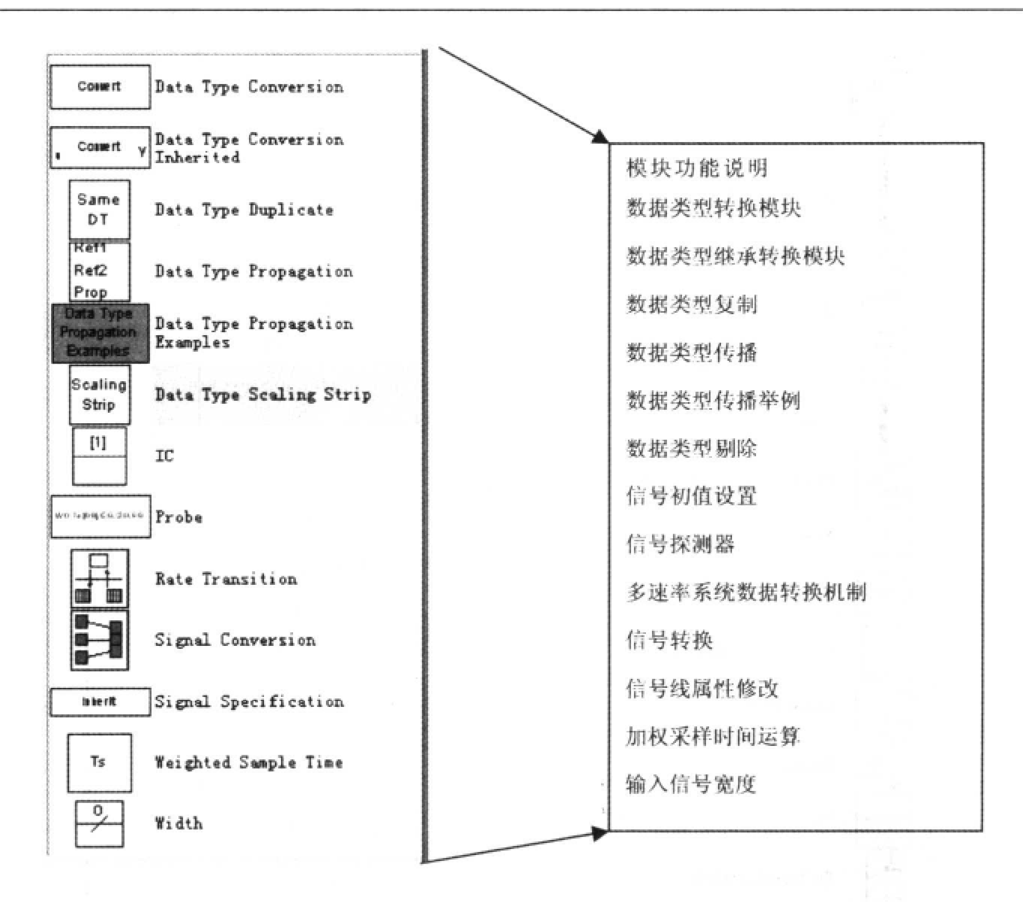


图 8.13　信号特性库及其模块功能

12. Signal Routing(*信号路由库*)

信号路由库及其各模块的功能如图 8.14 所示。

13. Sinks(*输出方式库*)

输出方式库及其各模块的功能如图 8.15 所示。

14. Source(*输入源*)

输入源库及其各模块的功能如图 8.16 所示。

15. User_Defined Function(*用户自定义函数库*)

用户自定义函数库及其各模块的功能如图 8.17 所示。

16. *其他数学离散模型库*

8.2.2　Simulink 的专业模型库

前面对 Simulink 的公共模型库做了详细的介绍，除了公共模型库外，Simulink 中还集成了许多面向不同专业的专业模型库，不同领域的系统设计师可以使用这些系统模块快速构建自己的系统模型，然后在此基础上进行系统的仿真、分析，从而完成设计任务。下面仅介绍几种控制工程师可能用到的专业模型库的主要功能。

1. *航空航天模型库*(Aerospace Blockset)

提供航空航天设计师常用的执行机构模块、空气动力模型、动画模块、环境仿真模块、三自由度和六自由度运动方程模块、风场、大气重力等环境模块、各种控制器模块、涡扇发动机模块、坐标和单位转换模块等。

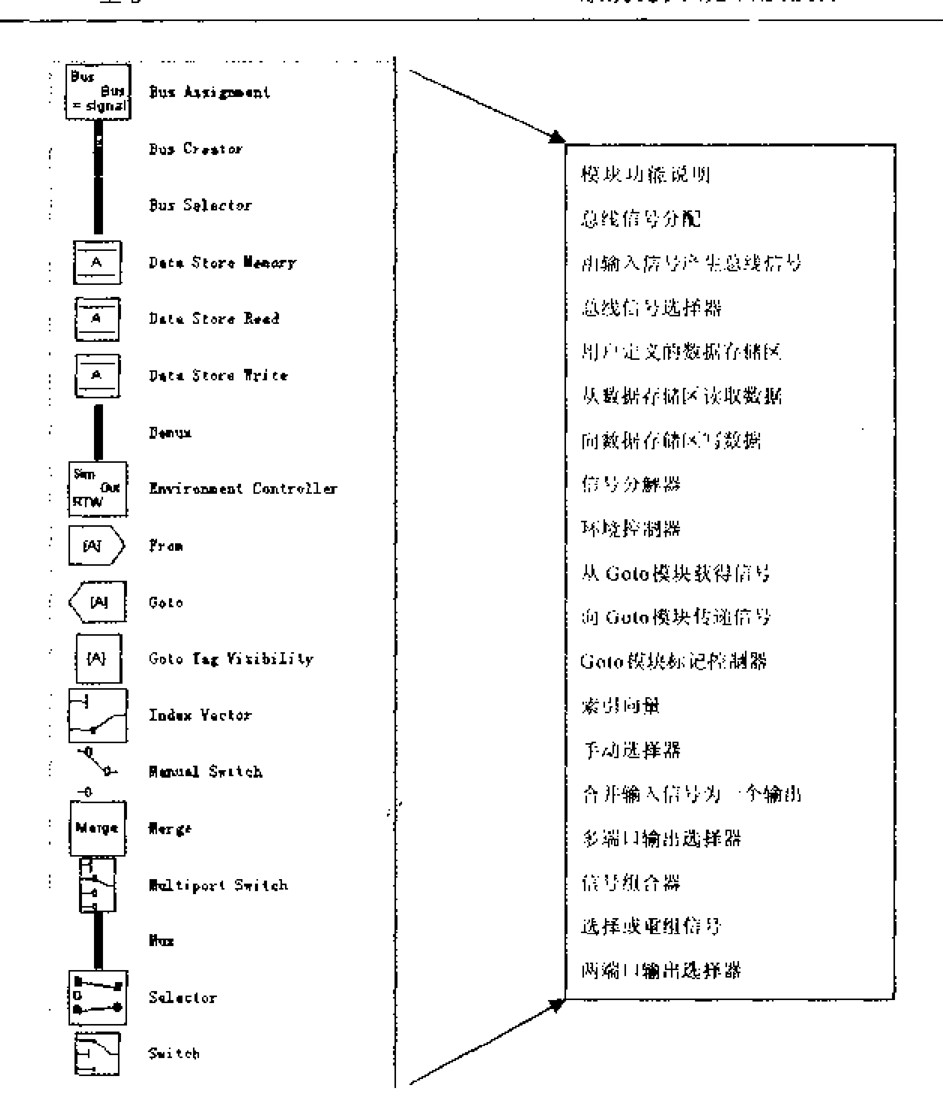


图 8.14 信号路由库及其模块功能

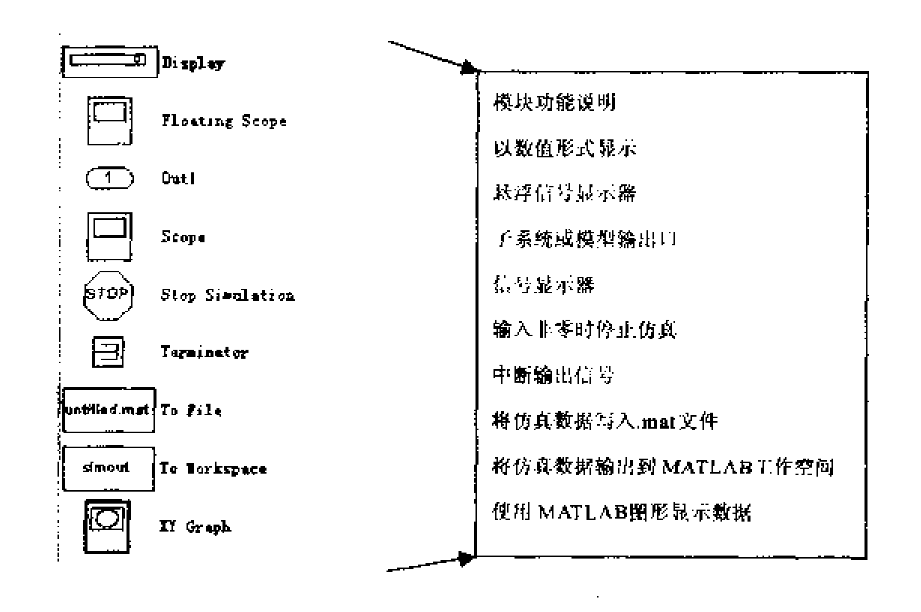


图 8.15 输出方式库及其模块功能

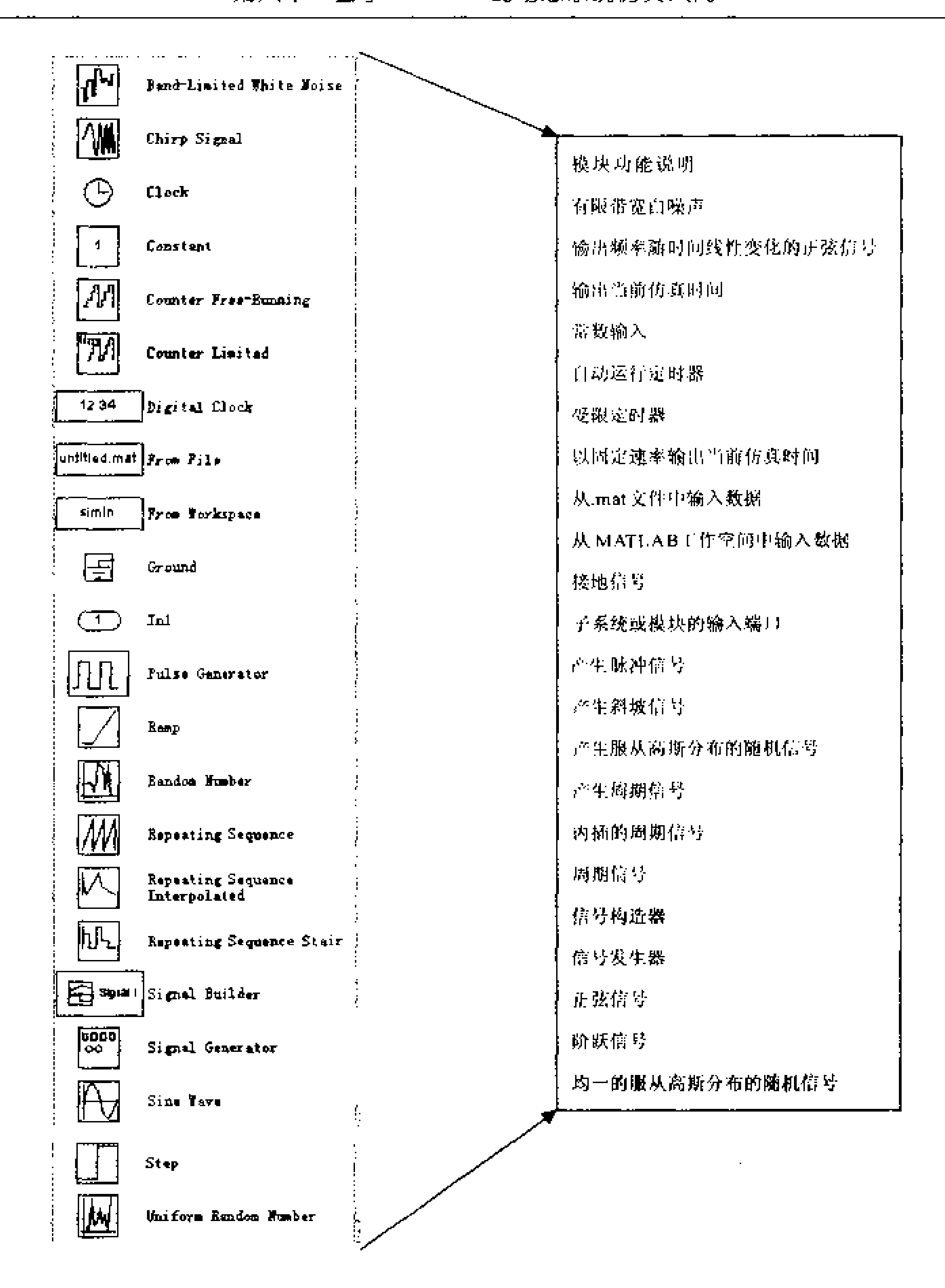


图 8.16　**输入源库及其模块功能**

2. 控制系统模型库(Control System Toolbox)

面向控制系统的设计和分析，主要提供线性时不变系统的模块。

3. 数字信号处理模型库(DSP Blockset)

面向数字信号处理系统的设计和分析，主要提供 DSP 输入/输出模块、信号预测与估计模块、滤波器模块、DSP 数学函数库、量化器模块、信号管理模块、信号操作模块、统计模块和信号变换模块等。

4. Simulink 附加模型库(Simulink Extras)

补充 Simulink 公共模型库，提供附加离散系统模型库、附加连续系统模型库、附加输出模型库、触发器模型库、线性化模型库和转换模型库。

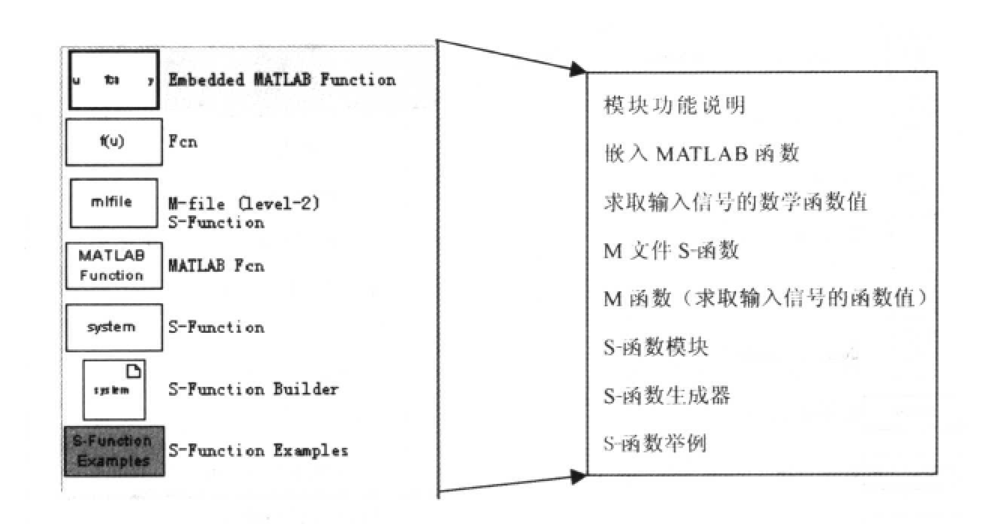


图 8.17 用户自定义函数库及其模块功能

5. S-函数示例模型库(S-function demos)

提供 C++,C,FORTRAN 以及 M 文件下的 S-函数模型库的演示模块。

6. 实时工作间、实时工作间嵌入式编码器及实时目标模型库(Real-Time Workshop,Real-Time Workshop Embedded Coder 和 Real-Time Windows Target)

提供各种用来进行独立可执行代码或嵌入式代码生成,以实现高效实时仿真的模块。它们和 RTW 及 TLC 有着密切的关系。

7. 状态流模型库(Stateflow)

对使用状态图所表达的有限状态机模型进行建模仿真和代码生成,有限状态机用来描述基于事件的控制逻辑,也可以用于描述响应型系统。

8. 通信模型库(Communication Blockset)

专用于通信系统仿真的一组模块。

9. 图形仪表模型库(Gauges Blockset)

实际上是一组 ActiveX 控件。

10. 神经网络模型库(Neural Network Blockset)

用于神经网络的分析设计和实现的一组模块。

11. 模糊控制模型库(Fuzzy Logic Toolbox)

用于模糊控制的分析设计和实现的一组模块。

12. 虚拟现实工具箱(Virtual Reality Toolbox)

提供进行虚拟现实仿真分析的各种工具,包括输入、输出、信号扩展器等。

13. xPC 模型库

提供一组用于 xPC 仿真的模块。xPC 是利用 PC 机,使用客户机服务器的模式进行实时仿真的一种经济仿真方案。它和 Simulink,RTW 相结合,可以在 PC 上进行单任务的实时仿真。

8.3 Simulink 模型的构建

8.3.1 对 Simulink 库浏览器的基本操作

第 8.2 节简单介绍了 Simulink 中的一些比较常用的系统模块。知道了这些模块的功能后，就可以使用这些系统模块构建用户自己的模型了。在构建用户模型的过程中，一定需要查找所需的模块，通过鼠标左键单击模型库的名称可以查看模型库中的模块，模型库中的模块会显示在 Simulink 库浏览器右边的一栏中。对 Simulink 库浏览器的基本操作主要包括：

(1) 使用鼠标左键单击模型库，则会在 Simulink 库浏览器右边的一栏中显示该库中的所有模块。

(2) 使用鼠标左键单击系统模块，则会在模块描述栏中显示该模块的功能描述。

(3) 使用鼠标右键单击系统模块，可以得到该模块的帮助信息，将模块拖进系统模型中，可以查看模块参数设置。

8.3.2 模块的基本操作

1. 模块的选择

假设用户需要将正弦信号通过信号显示器显示。这时就需要两个模块：由系统输入模型库 Sources 中的 Sine Wave 模块产生正弦信号，用系统输出模型库 Sinks 中的 Scope 模块显示结果。

启动 Simulink 并新建一个系统模型文件。选择上面提到的两个模块将其拷贝(或拖)到新建的系统模型中，如图 8.18 所示。

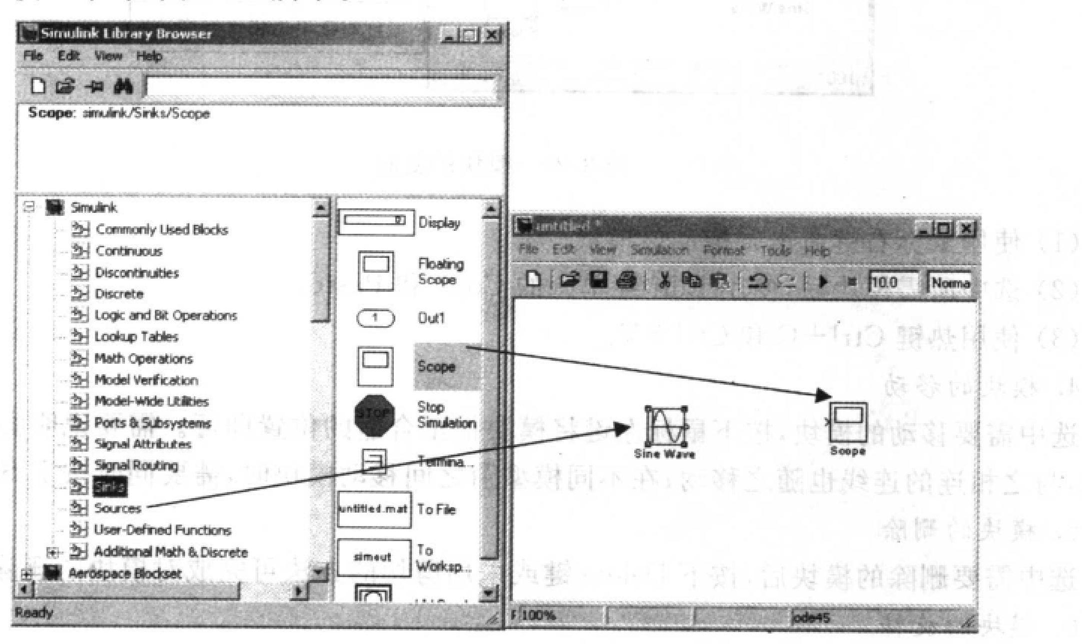


图 8.18 模块选择示例

2. 模块的连接

选择好构建系统所需的模块后，需要按照系统的信号流程将各个模块正确连接。将光标指向起始块的输出端口，此时光标变成“+”。单击鼠标左键并拖动到目标模块的输入端口，在接近到一定程度时光标变成双“+”，此时松开鼠标键即可。完成后在连接点处出现一个箭头，表示系统中信号的流向，如图 8.19 所示。

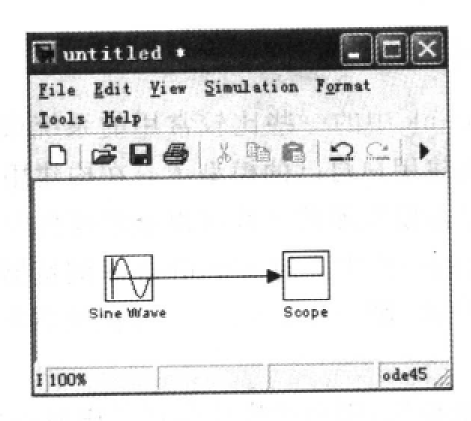


图 8.19　系统模块之间的连线

3. 模块的复制

如果需要几个相同的模块，可以使用 3 种方法进行复制，结果如图 8.20 所示。

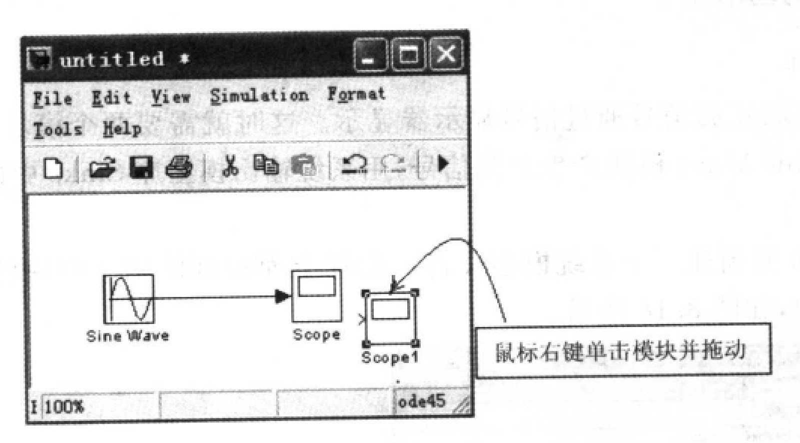


图 8.20　模块的复制

(1) 使用鼠标右键单击并拖动该模块。

(2) 选中所需模块后，使用 Edit 菜单上的 Copy 和 Paste。

(3) 使用热键 Ctrl+C 和 Ctrl+V。

4. 模块的移动

选中需要移动的模块，按下鼠标左键将模块拖至合适的位置即可。需要说明的是，模块移动时，与之相连的连线也随之移动；在不同模型窗之间移动模块时，需要同时按下 Shift 键。

5. 模块的删除

选中需要删除的模块后，按下 Delete 键或采用剪切的方法可完成对模块的删除。

6. 模块的旋转

在构建仿真模型时，有时系统的信号并非都是从左到右的，因而模块不能总处于其缺省的

输入端在左，输出端在右的状态。在选中需要旋转的模块后，有两种方法可以旋转模块：

(1) 使用 Ctrl+R 热键；

(2) 使用 Format 菜单下的 Flip Block 可将模块旋转 180°，Format 菜单下的 Rotate Block 可将模块旋转 90°。

7. 模块名的操作

(1) 修改模块名。点击模块名，会在原名字四周出现一个编辑框。此时可对模块名进行修改。修改完毕，将光标移出该编辑框，点击即结束修改。

(2) 模块名字体设置。使用 Format 菜单下的 font，打开字体设置对话框进行设置。

(3) 改变模块名的位置。选中模块后，使用菜单 Format 下的 Flip Name，可将模块名移至模块对面或利用鼠标拖动模块名编辑框移至对面。

(4) 隐藏模块名。选中模块后，选用 Format 菜单下的 Hide Name 即可隐藏模块名，同时，菜单也变为 Show Name。

8. 模块的阴影效果

使用 Format 菜单下的 Show Drop Shadow 可以给选定的模块加上阴影效果。

9. 模块颜色的改变

使用鼠标右键点击模块，选择 Foreground color 或 Background color 菜单来设置颜色；或使用 Format 菜单下的相应命令设置颜色。如果模块的前景色发生变化，则所有由此模块引出的信号线的颜色也随之发生改变。

10. 模块的插入

假设用户需要将输入信号放大后才显示，此时需要在图 8.19 所示的两个模块之间插入一个放大环节。用户只需将所需模块移到连接线上即可完成插入工作，如图 8.21 所示。

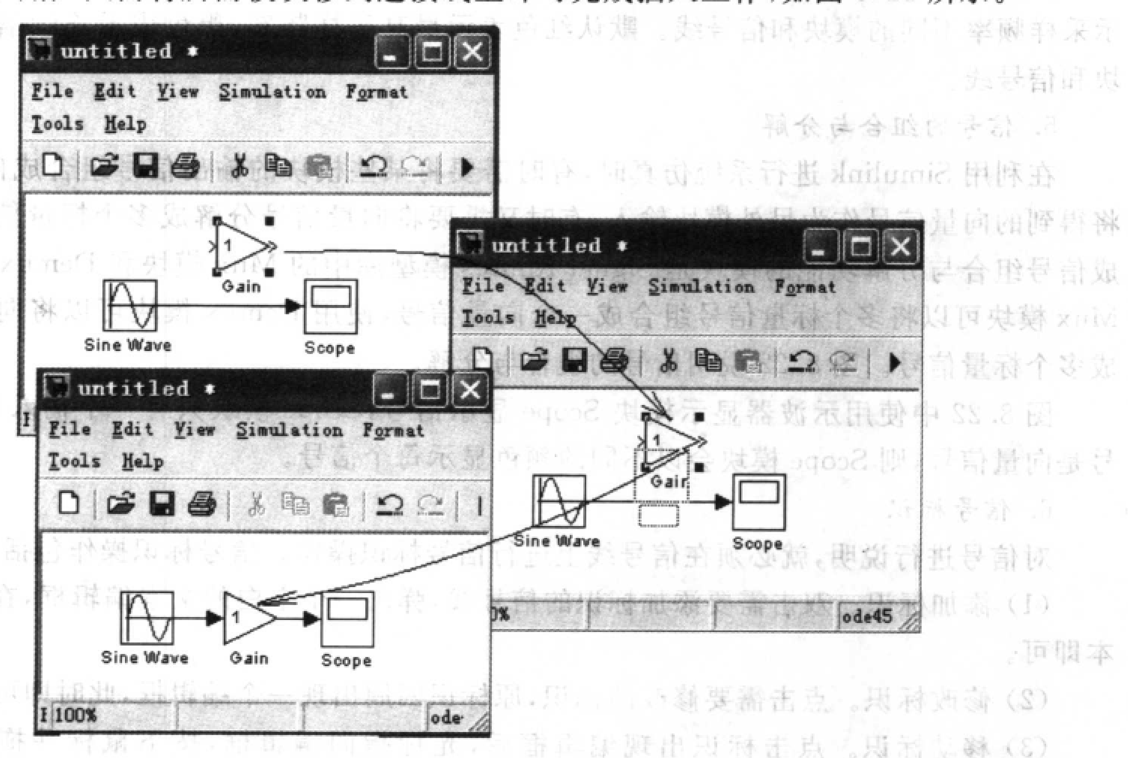


图 8.21 模块的插入

8.3.3 信号的操作

Simulink 模型中的信号总是由模块之间的连线携带并传送的，因此模块间的连线称为信号线。

1. 信号线的分支

有些系统模块的输出必须同时作为多个模块的输入，这时需要从此模块中引出若干连线，称作信号线分支。对信号线进行分支的操作有下列方式：

(1) 使用鼠标右键单击需要分支的信号连线拖至目标模块。

(2) 按下 Ctrl 键的同时，使用鼠标左键单击需要分支的信号连线拖至目标模块。

2. 信号线的曲折

对信号连线的路径的改变方式有：

(1) 使用鼠标左键单击并拖动；

(2) 按下 Shift 键的同时，使用鼠标左键单击并拖动，可以改变连线路径。

3. 信号线宽度的设置

信号线所携带的信息既可能是标量也可能是向量，并且不同信号线所携带的向量信号的长度可能互不相同。为了使信号传递一目了然，Simulink 不仅可以用粗宽线显示向量信号线，还可将向量的长度用数字标出。使用 Format 菜单中的 Port/Signal Display-Wide nonscale Lines 和 Signal Dimensions 完成这项功能。

4. 信号线的彩色显示

Simulink 所建的离散系统允许有多个采样频率。为了清楚显示不同采样频率的模块及信号线，选中 Format 菜单下的 Port/Signal Display-Sample Time Colors 可以用不同的颜色显示采样频率不同的模块和信号线。默认红色表示最高采样频率，黑色表示连续信号流经的模块和信号线。

5. 信号的组合与分解

在利用 Simulink 进行系统仿真时，有时需要将某些模块的输出信号组合成向量信号，并将得到的向量信号作为另外模块输入；有时又需要将向量信号分解成多个标量信号。能够完成信号组合与分解功能的模块是 Signal Routes 模型库中的 Mux 模块和 Demux 模块。使用 Mux 模块可以将多个标量信号组合成一个向量信号，使用 Demux 模块可以将向量信号分解成多个标量信号。图 8.22 说明信号的组合与分解。

图 8.22 中使用示波器显示模块 Scope 显示信号，Scope 模块只有一个输入口，若输入信号是向量信号，则 Scope 模块会以不同的颜色显示每个信号。

6. 信号标识

对信号进行说明，就必须在信号线上进行信号标识操作。信号标识操作包括：

(1) 添加标识。双击需要添加标识的信号线，弹出一个空白的文本编辑框，在其中输入文本即可。

(2) 修改标识。点击需要修改的标识，原标识四周出现一个编辑框，此时即可修改。

(3) 移动标识。点击标识出现编辑框后，光标指向编辑框，按下鼠标并拖至新的位置即可。

(4) 复制标识。与移动标识类似，只需同时按下 Ctrl 键或利用鼠标右键操作。

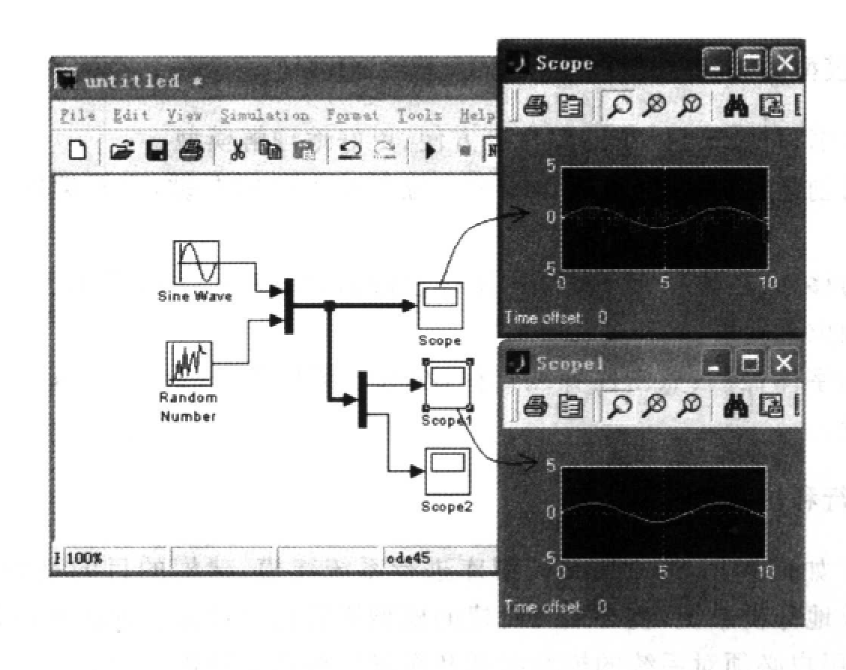


图 8.22　信号的组合与分解

(5) 删除标识。点击标识出现编辑框后，双击标识选中整个标识，按下 Delete 键即可删除标识。

(6) 标识的传递。在 Simulink 模型库中，有一些像 Demux，Mux，Goto，From 等模块具有传递信号标识的功能。这种功能可使整个模型中的信号传递路径清晰，便于分析检查。信号标识的传递方法有两种。第一种方法，选择信号线并用鼠标左键双击，在信号标识编辑框中键入"＜""＞"，在此尖括号中键入信号标签即可传递信号标识。然后选择 Edit 菜单下的 Update Diagram 刷新模型。第二种方法，先选择信号线，然后选择 Edit 菜单中的 Signal Properties；或单击鼠标右键，选择弹出式菜单中的 Signal Properties，将 Show Propagated Signals 设置为 on 即可。图 8.23 说明了信号标识的传递过程。

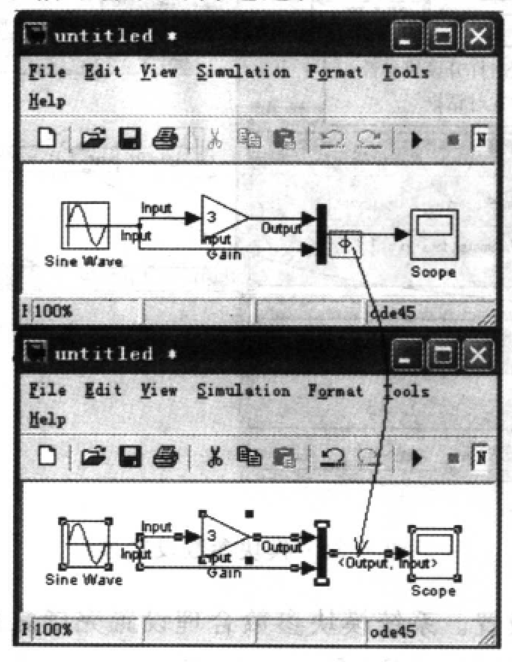


图 8.23　信号标识的传递

8.3.4 对模型的注释

在模型窗中书写注释可以帮助用户更方便、更好地理解模型。

模型注释的创建：在将用做注释区的中心位置双击鼠标左键，出现编辑框，在框中输入所需的文字即可。

注释位置的移动：在注释文字处单击鼠标左键出现编辑框，将光标移至编辑框上按下鼠标左键并拖至希望位置即可。

注释文字的字体控制：点击注释编辑框，在选中菜单 Format 下的 Font，弹出标准的字体对话框进行设置。

8.3.5 运行和仿真

前面讲述了如何利用 Simulink 模型库进行系统建模，建模的目的是为了分析和设计系统。为了能有效地分析系统，就必须对所建的模型进行仿真计算。现在讨论运行仿真的问题。在系统仿真前，用户必须对系统的模块参数和系统的仿真参数进行设置。

1. 系统模块参数设置与系统仿真参数设置

在用户按照信号的输入输出关系连接好各个系统模块之后即完成了模块的构建工作。为了正确仿真和分析必须正确设置系统模块参数和仿真参数。

(1) 系统模块参数的设置。双击系统模块打开包含该模块的简单描述和模块参数选项等信息的系统模块对话框，在该参数对话框中正确设置参数即可。模块参数设置过程如图 8.24 所示。

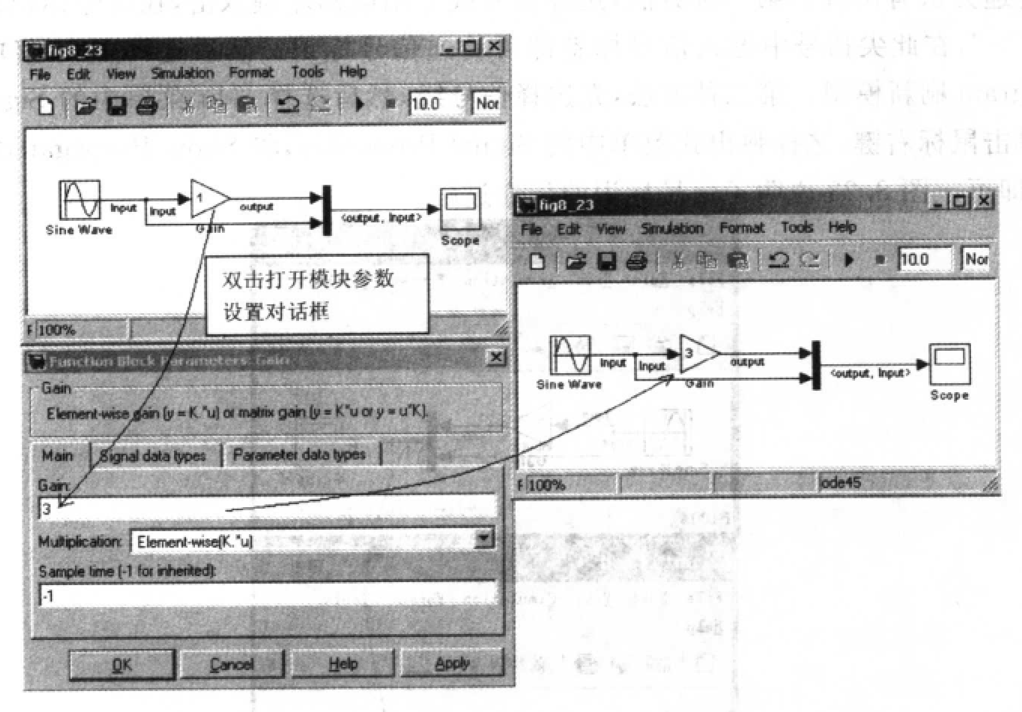


图 8.24 模块参数设置过程

(2) 系统仿真参数的设置。系统模块参数合理设置完毕之后就可设置系统仿真参数了，Simulink 的窗口中的 simulation / Configuration Parameters 选项就是用来设置仿真参数的。

选中该选项后，弹出仿真参数设置对话框，如图 8.25 所示。其中包括了 7 个选择页，它们是 Solver(求解器)，Data Import/Export(数据输入/输出页)，Optimization(优化页)，Diagnostics(诊断页)，Hardware Implementation(硬件实现页)，Model Referencing(参考模型页)，Real-time workshop(实时工作间页)。这里主要讲述 Solver 页，因为求解器的正确设置是得到合理、可信的仿真结果的前提条件，所以有必要详细了解求解器的设置内容和设置方法，以便进行正确的设置。关于 Data Import/Export(数据输入/输出页)，Diagnostics(诊断页)，Real-time workshop(实时工作间页)的设置与功能，在后面的章节中再做介绍。

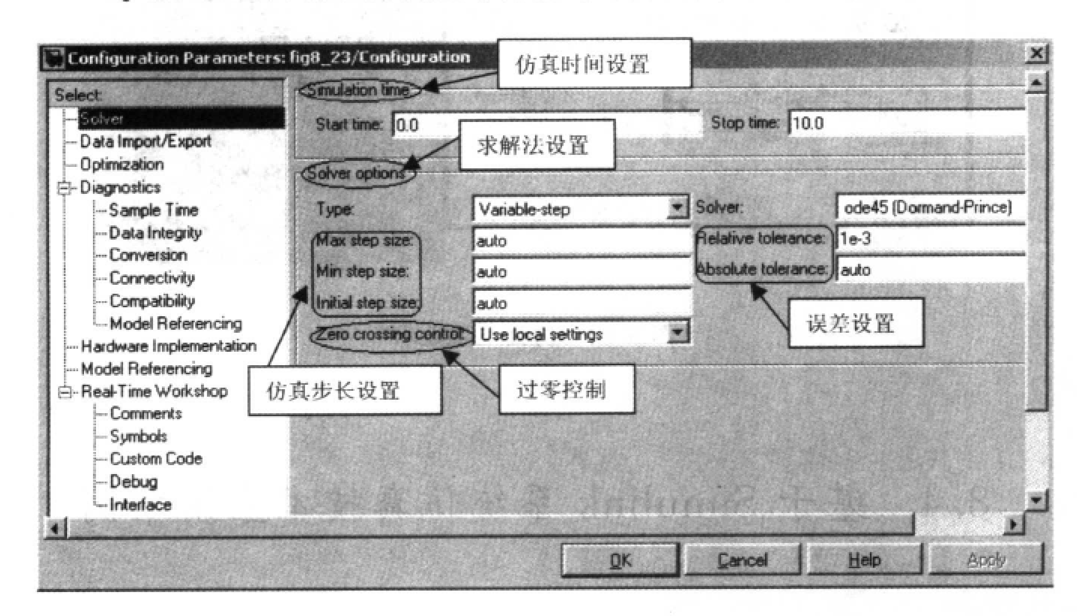


图 8.25 仿真参数设置对话框

通过 Simulink 的窗口中的 Simulation / Configuration Parameter 打开的对话框停留在 Solver 求解器设置页，如图 8.25 所示。Solver 页包含了 simulation time(设置仿真时间)和 solver options(选择求解法)。这两个选项又各自包含若干小选项：

Simulation time / Start time 仿真起始时间(秒)

Stop time 仿真终止时间(秒)

Solver options

Type /variable step 变步长

Fixed step 固定步长

Ode45，ode23 常微分方程的 runge-kutta 解法等选择

Max step size 最大步长

Min step size 最小步长

Initial step size 初始步长

Relative tolerance 相对误差

Absolute tolerance 绝对误差

Zero crossing control 仿真时过零控制

对应图 8.24 所示系统，系统仿真参数采用 Simulink 中默认的设置。

2. 运行仿真

完成系统模块的构建、模块参数设置和仿真参数设置后，即可进行系统仿真了。单击模型文件工具栏上的 Start Simulation 图标 ▶ 或通过 Simulation 菜单下的 Start 来启动仿真。仿真结束后，双击 Scope 模块可以显示仿真结果，如图 8.26 所示。

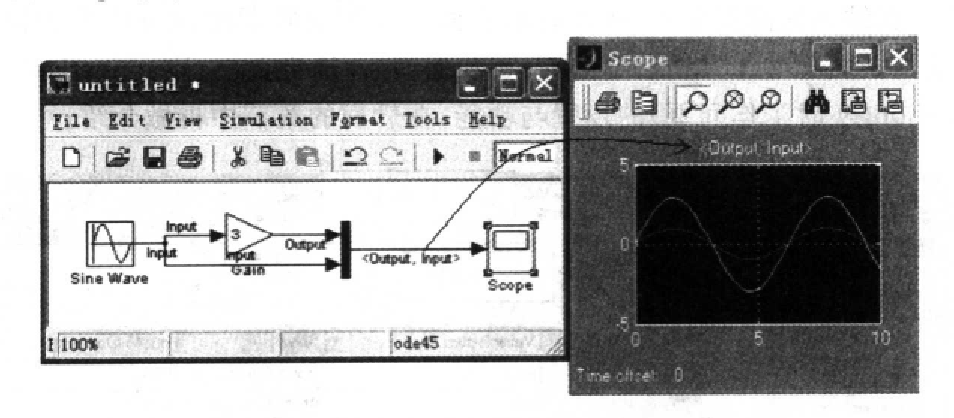


图 8.26　仿真结果显示

8.4　基于 Simulink 系统仿真技术应用举例

用 Simulink 建模并仿真应按照四个步骤进行，包括建模、设置模型参数、设置仿真参数和进行仿真分析。本节通过对连续系统、离散系统和混合系统的举例，使读者更牢固地掌握系统建模仿真的步骤、模型参数、仿真参数的设置方法以及仿真结果的分析和显示。

8.4.1　连续系统仿真分析

连续系统中既包含连续的线性系统也包含连续的非线性系统。首先举一个连续线性系统的例子，说明如何使用 Simulink 对连续线性系统进行仿真分析。

例 8.1　单位反馈的二阶系统如图 8.27 所示。因为该系统的阻尼比较小，实际使用时需进行性能改善。采用比例加微分控制，可以在系统出现位置误差之前，提前对系统产生修正作用，最终达到改善系统性能的目的。加入比例加微分控制后的系统模型图及其仿真结果如图 8.28 所示。

解

(1) 模型创建。建立本系统模型需用的模块有：

Source 模型库中的 Step 模块：输入阶跃信号；

Math Operations 中的 Gain 和 Sum 模块；

Continuous 中的 Derivatives 和 Zero-Pole 模块；

Sinks 中的 Scope 模块。

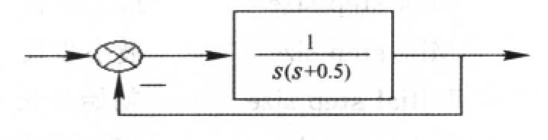


图 8.27　单位反馈二阶系统

(2) 模型参数及系统仿真参数设置。

1) 系统模块参数设置：

Zero-Pole 模块设置：将零点 Zeros、极点 Poles 和增益分别设置为[ ]，[0　0.5]和 1；

Step 模块设置：使用系统的默认取值，即 1s 时阶跃的单位阶跃信号；

其他模块设置如图 8.28 所示。

2）系统仿真参数设置：使用 Simulation parameters 仿真参数对话框中的 Solver 选项卡设置参数。

仿真时间范围设置为 0～20s；

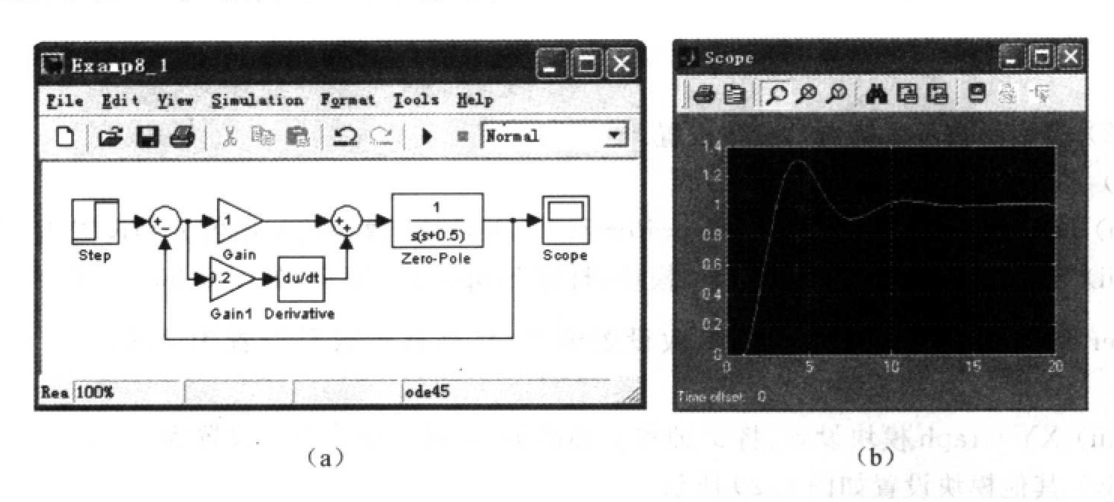
使用变步长连续求解器(variable - step)，仿真算法为 ode45；

最大仿真步长(Max step size)为 0.01；

绝对误差(Absolute tolerance)为 1e−6；

其余仿真参数使用默认值。

(3) 仿真。对模块参数和仿真参数进行了合理的设置之后，可以进行系统仿真，点击模型文件工具栏上的 Start Simulation 图标▶或通过 Simulation 菜单下的 Start 来启动仿真。仿真结束后，双击 Scope 模块可以显示仿真结果，如图 8.28 所示。



(a)　　(b)

图 8.28　比例加微分控制系统模型及其仿真结果

例 8.2　连续的非线性系统举例。利用 Simulink 计算 Van der pol 方程：

$$\begin{cases}\dot{x}_1 = x_2 \\ \dot{x}_2 = -m(x_1^2 - 1)x_2 - x_1\end{cases}$$

并用示波器 Scope 显示状态量 x_1 和 x_2。

解

(1) 系统模型创建。建立本系统模型需用的模块有：

Math Operations 中的 Gain，Slider Gain，Product 和 Sum ；

Source 中的 Constant 模块；

Continuous 中的 Integrator 模块；

Signal Routing 中的 Mux 模块：将两路输入进行向量化，混合成一路输出，直接连接到示波器上，可同时绘制两条曲线；

Sinks 中的 Scope 模块和 XY Graph 模块。由 Scope 模块显示系统各状态的时间响应曲线，XY Graph 显示系统的相平面曲线。

按照图 8.29 构建系统 Simulink 模型。

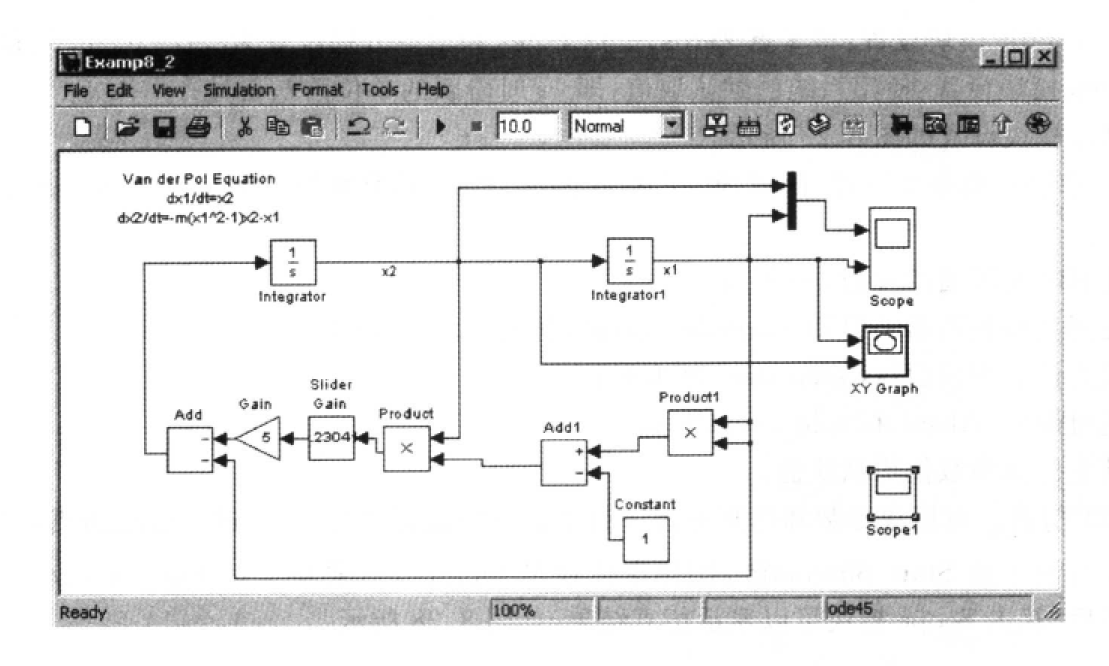


图 8.29 求解 Van der Pol 方程的 Simulink 模型

(2) 模型参数及系统仿真参数设置。

1) 系统模块参数设置：

(i) Integrator 模块设置：将 Integrator 和 Integrator1 模块的初始值分别设置为 −2 和 1；

(ii) Scope 模块设置：双击 Scope 模块，打开 Scope 显示窗口，如图 8.30(a)所示，点击 Parameter 图标 即可打开 Scope 参数设置窗口，将坐标轴数目设置为 2，即可产生两个示波器。

(iii) XY Graph 模块设置：将 x 轴和 y 轴的最小、最大值均分别设置为 −5 和 5。

(iv) 其他模块设置如图 8.29 所示。

2) 仿真参数的设置：均使用 Simulink 的默认值。

点击仿真开始图标 ▶，启动仿真程序，仿真结束后，系统的仿真结果可由 Scope 和 XY Graph 模块观察，如图 8.30(b)和(c)所示。

8.4.2 离散系统仿真分析

许多离散系统中包含多种不同的采样速率。通常在离散控制系统中，控制器的更新速率一般要低于对象本身的工作频率，显示系统的更新速率更是比显示器的可读速度低得多。

例 8.3 假设某离散系统的状态方程为

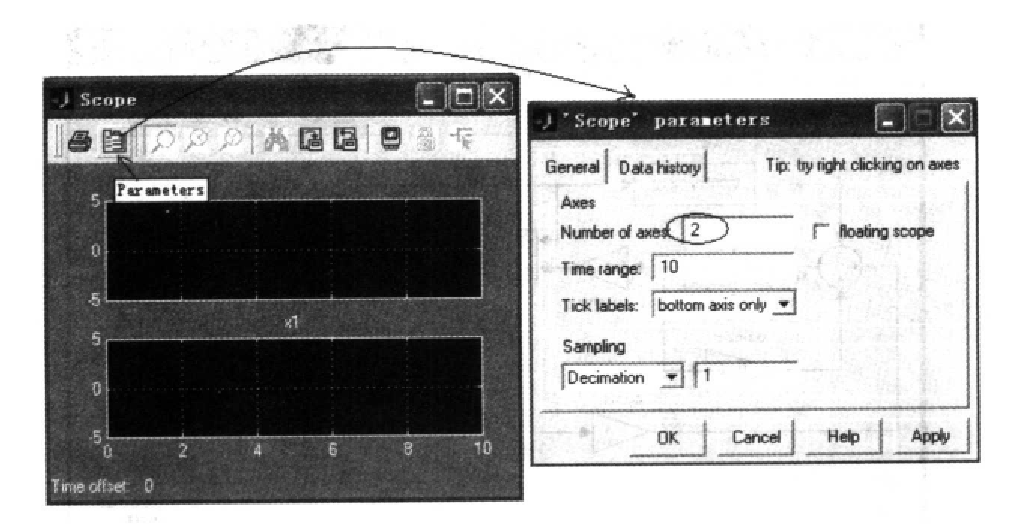
$$\begin{cases} x_1(k+1) = x_1(k) + 0.1x_2(k) \\ x_2(k+1) = -0.05\sin(x_1(k)) + 0.094x_2(k) + u(k) \end{cases}$$

其中 $u(k)$ 是输入，$u(k) = 0.75 - x_1(k)$ 。该过程的采样周期是 0.1 s。控制器采样周期为 0.25 s，显示系统的更新周期为 0.5 s。

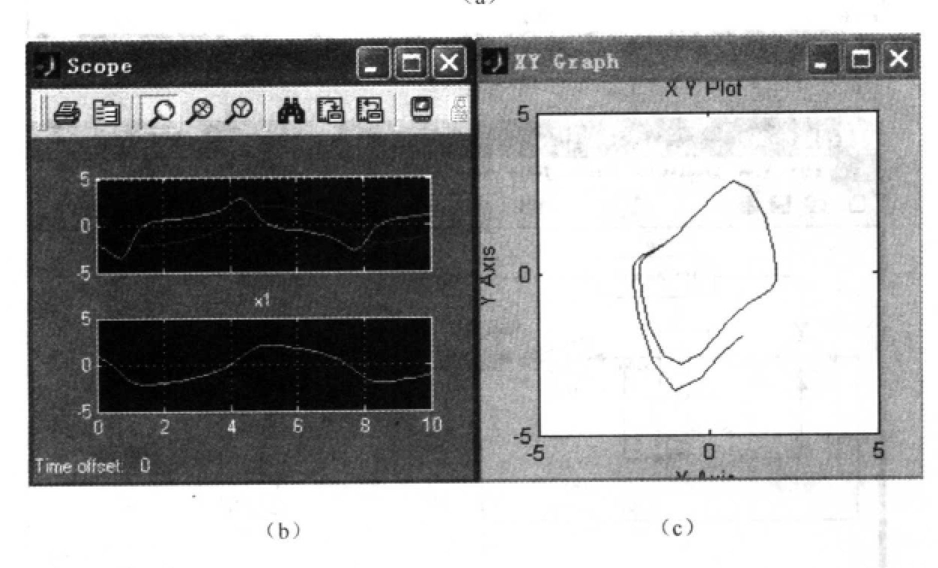
解

(1) 系统模型创建，如图 8.31 所示，需用的模块有：

Math Operations 中的 Gain 和 Sum；



(a)



(b)　(c)

图 8.30　Van der Pol 方程求解的 Scope 模块设置及仿真结果

(a) Scope 模块设置；(b) 时间响应曲线；(c) 相平面曲线

Source 中的 Constant 模块；

Discrete 中的 Unite Delay 和 Zero-Order Hold 模块；

Sinks 中的 Scope 和 Display 模块；

User-Defined Function 中的 Fcn 模块。

(2) 系统模块参数及仿真参数设置。

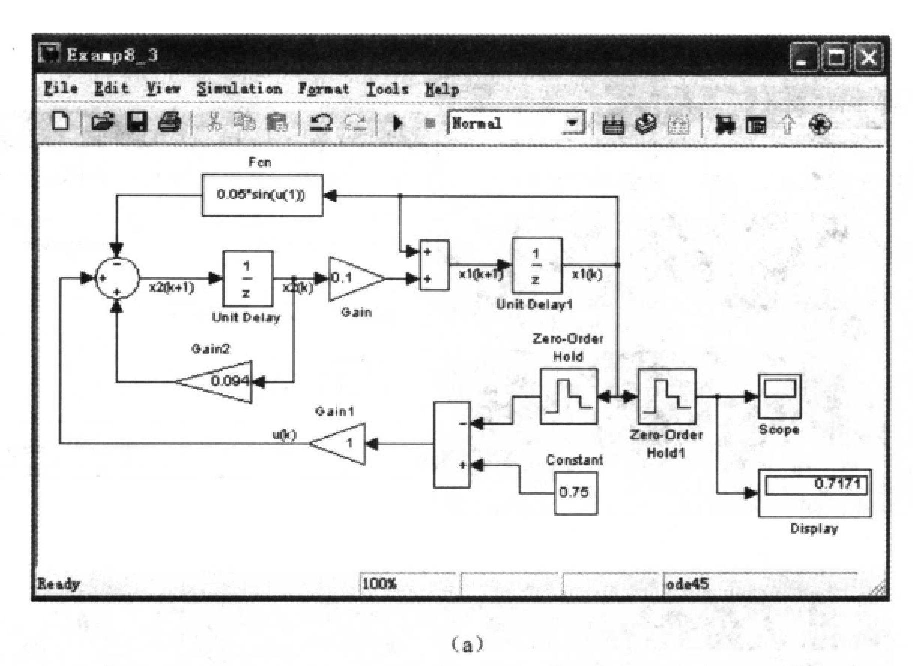
1) 系统模块参数设置：

(i) Unite Delay 和 Unite Delay1 模块设置：双击模块，打开模块参数窗口，将此两个模块的采样时间均设置为 0.1 s；

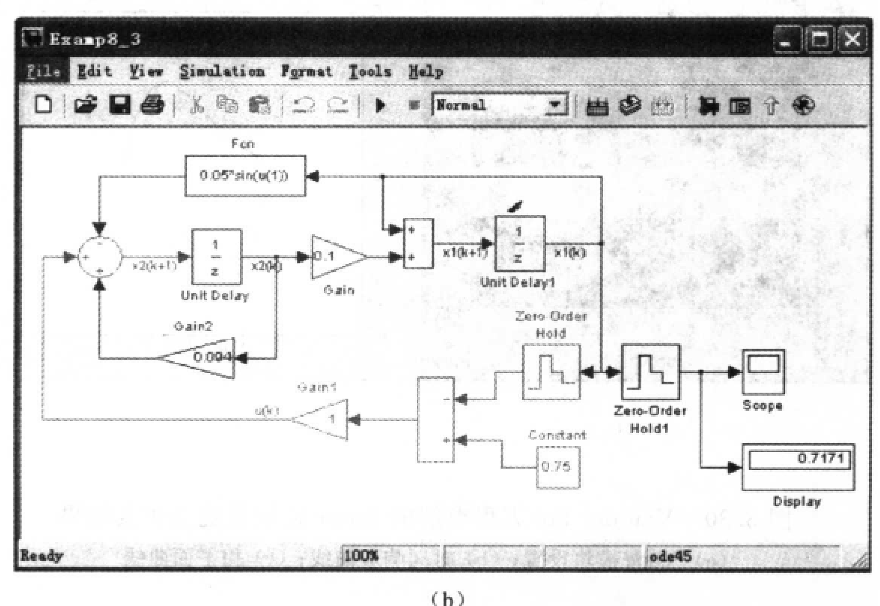
(ii) Zero-Order Hold 和 Zero-Order Hold1 模块设置：双击模块，打开模块参数窗口，将 Zero-Order Hold 和 Zero-Order Hold1 模块的采样时间分别设置为 0.25 s 和 0.5 s；

(iii) Fcn 模块设置：双击 Fcn 模块，将其表达式设置为 0.05 * sin(u(1))；

其他模块设置如图 8.31(a)所示。



(a)



(b)

图 8.31 例 8.3 离散系统仿真模型

(a) 未加注采样周期的仿真模型； (b) 加注不同采样周期的仿真模型

2) 仿真参数的设置。求解器设置为离散求解器，其余均参用 Simulink 的默认值，如图 8.32 所示。

在多采样周期的复杂系统中，为了分清各部分信号的采样周期，可以用不同的颜色标记不同采样周期的信号。具体的方法是在 format 菜单下，点击 Port/Signal Displays-Sample time colors 即可，如图 8.31(b)所示。

(3) 仿真。将系统模块参数与系统仿真参数设置之后，点击仿真开始图标▶，启动仿真程序。在仿真过程中，Display 模块实时显示 $x_1(k)$的数值，$x_1(k)$的历史记录，可以由 Scope 模块观察，如图 8.33 所示。

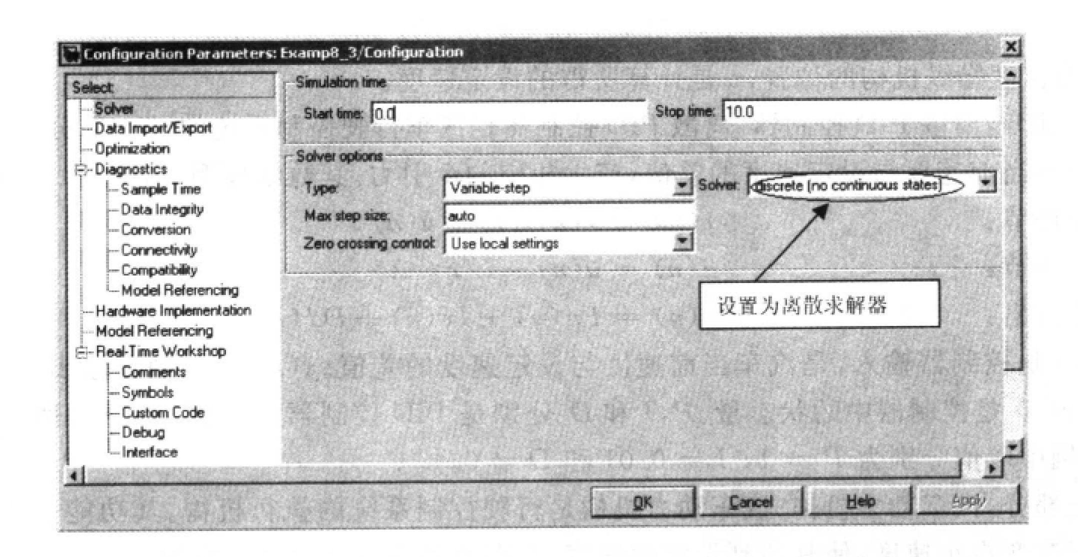


图 8.32　离散系统仿真参数设置

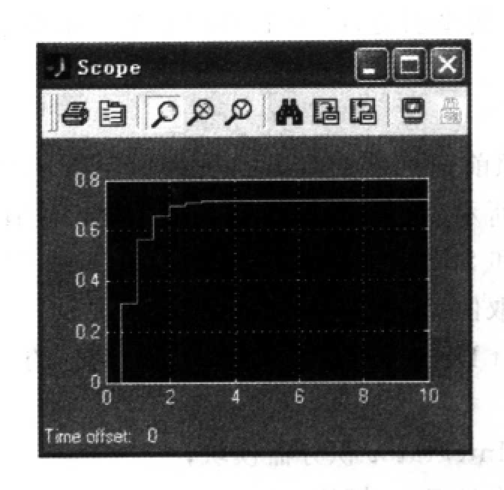


图 8.33　例 8.3 离散系统仿真结果

8.4.3　混合系统仿真分析

实际的系统常常是混合系统(即系统中有连续信号也有离散信号)。在对这类系统进行仿真时,必须考虑连续信号和离散信号采样时间之间的匹配问题。Simulink 中的变步长连续求解器充分考虑了上述问题。所以在对混合系统进行仿真分析时,应该使用变步长连续求解器。

例 8.4　汽车行驶控制系统是应用很广的控制系统之一,控制的目的是对汽车速度进行合理的控制。它是一个典型的反馈控制系统,其工作原理如下:

使用汽车速度操纵机构的位置变化量设置汽车的指定速度,这是因为操纵机构的不同位置对应着不同的速度;测量汽车的当前速度,求取它与指定速度的差值;由差值信号产生控制信号驱动汽车产生相应的牵引力以改变并控制汽车速度直到达到指定速度。在对这个系统进行建模仿真前,需要先对此系统做简单的介绍。

汽车行驶控制系统包含三部分机构。

第一部分,速度操纵机构的位置变换器。位置变换器是汽车行驶控制系统的输入,其作用是将速度操纵机构的位置转换为相应的速度。速度操纵机构的位置和设定速度间的关系为

$$v = 50x + 45 \quad x \in [0,\ 1]$$

式中，x 为速度操纵机构的位置，v 是计算所得的设定速度。

第二部分，离散 PID 控制器。离散 PID 控制器是汽车行驶控制系统的核心部分。其作用在于根据汽车当前速度与设定速度的差值，产生相应的牵引力。其数学模型为：

积分环节：　$x(n) = x(n-1) + u(n)$

微分环节：　$d(n) = u(n) - u(n-1)$

系统输出：　$y(n) = Pu(n) + Ix(n) + Dd(n)$

其中 $u(n)$ 是控制器输入，是汽车当前速度与设定速度的差值。$y(n)$ 是控制器输出，即汽车的牵引力，$x(n)$ 是控制器中的状态量。P，I 和 D 分别是 PID 控制器的比例、积分和微分控制参数，在本例中取值分别为 $P = 1$，$I = 0.01$ 和 $D = 0$。

第三部分，汽车动力机构。汽车动力机构是行驶控制系统的执行机构。其功能是在牵引力的作用下改变汽车速度，使其达到设定的速度。牵引力与速度之间的关系为

$$F = m\dot{v} + bv$$

式中，v 是汽车速度，F 是汽车的牵引力，$m = 1\ 000$ kg 是汽车质量，$b = 20\ \mathrm{N \cdot s \cdot m^{-1}}$ 是阻力因子。

解

(1) 系统模型创建。按照前面给出的汽车行驶控制系统的数学模型，构建系统的 Simulink 仿真模型，如图 8.34(a)所示。此仿真模型需要的系统模块有：

Math 模块库中的 Gain 和 Slider Gain 滑动增益模块：Slider Gain 滑动增益模块用来调节位置变换器的输入信号 x 的取值；

Discrete 模型库中的 Unit Delay 单位延迟模块：产生信号的一步延迟，以实现 PID 控制算法；

Continuous 模型库中的 Integrator 积分器模块；

Math Operation 模型库中的 Sum 模块。

(2) 系统模块参数及仿真参数设置。

1) 系统模块参数设置：

Slider Gain 模块：最小值 Low 为 0，最大值 High 是 1，可取 0～1 之间的任意值；

Unit Delay 模块：初始状态为 0，采样时间为 0.02s；

Intergrator 模块：初始状态为 0。

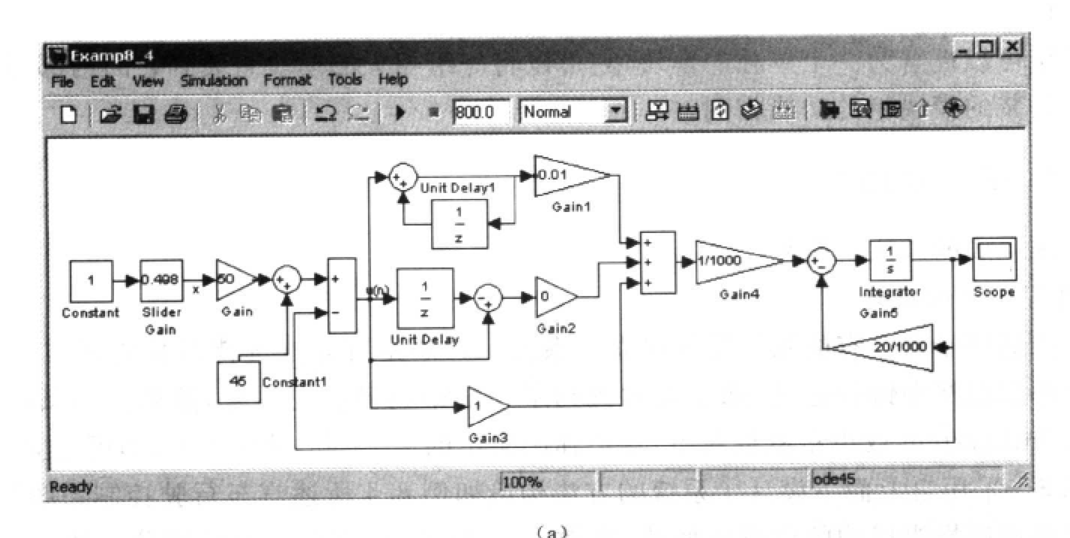
其余模块的参数设置参见系统仿真模型图 8.34(a)或使用默认取值。

2) 系统仿真参数设置：

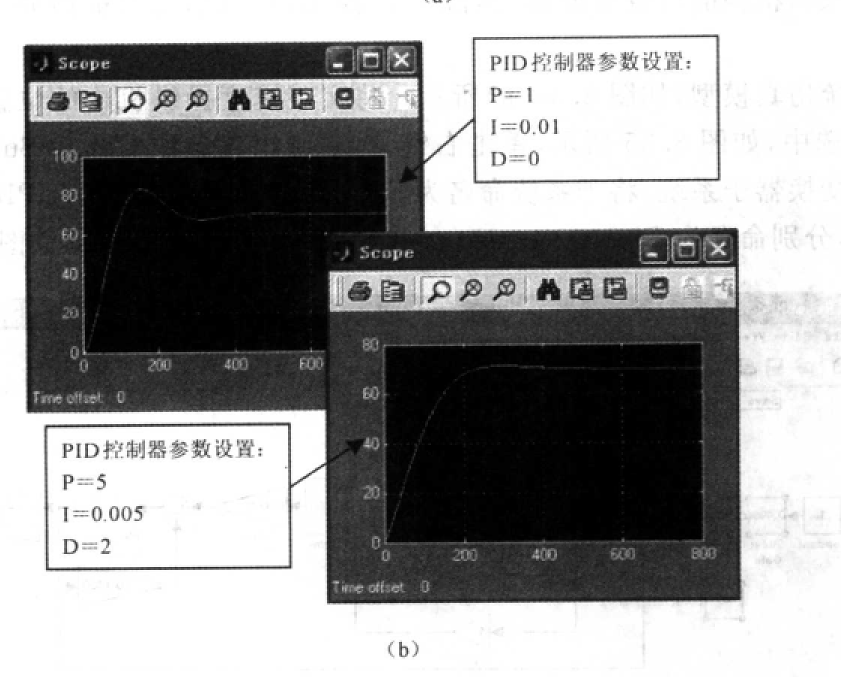
仿真时间范围：0～800 s；

求解器：使用变步长连续求解器。

(3)系统仿真与分析。将系统模块参数与系统仿真参数设置之后，对系统进行仿真，系统的仿真结果如图 8.34(b)所示。



(a)



(b)

图 8.34　汽车行驶控制系统仿真模型及其仿真结果

(a) 汽车行驶控制系统仿真模型；（b）汽车行驶控制系统仿真结果

8.5　Simulink 的子系统技术

对于简单的系统，可以直接使用前面介绍的方法建立 Simulink 仿真模型进行动态系统仿真。然而，对于复杂的动态系统，直接对系统进行建模，不论是分析系统还是设计系统，都会给用户带来极大的不便。对于复杂系统，因为其动态系统中包含的功能模块较多，模块间的输入、输出关系比较复杂，因而应该采用适当的策略建立系统的模型。Simulink 的子系统技术可以较好地解决复杂系统的建模、仿真问题。这是因为模型中的子系统可以大大地增强系统

模型的可读性。

子系统可以理解为一个单独的模块，它可以将一组相关的模块封装到它的内部，其实现的功能与其封装的模块组的功能相同。

8.5.1 子系统的生成

子系统生成的方法有两种：

1. 自下而上的设计

此方法适用于对已有的系统模型建立子系统。方法是首先框选待封装的区域，即在模型编辑窗中单击鼠标左键并拖动，选中需放置到子系统中的模块与信号，然后选择 Edit 菜单下的 Create Subsystem 或单击鼠标右键，选中弹出菜单的 Create Subsystem，即可建立子系统。

例 8.5 采用自上而下设计子系统的方法构造如例 8.4 所述汽车行驶控制系统所示的模型。要求将速度操纵机构的位置变换器、离散 PID 控制器、汽车动力机构分别用不同的子系统表示。

首先建立系统仿真模型，如图 8.34(a)所示。将实现速度操纵机构的位置变换器功能的各个模块和信号选中，如图 8.35 所示，单击右键，选中弹出菜单的 Create Subsystem 建立速度操纵机构位置变换器子系统，将子系统命名为 Set Speed。同法建立离散 PID 控制器和汽车动力机构子系统，分别命名为 Discrete cruise controller 和 Car dynamics，如图 8.36 所示。

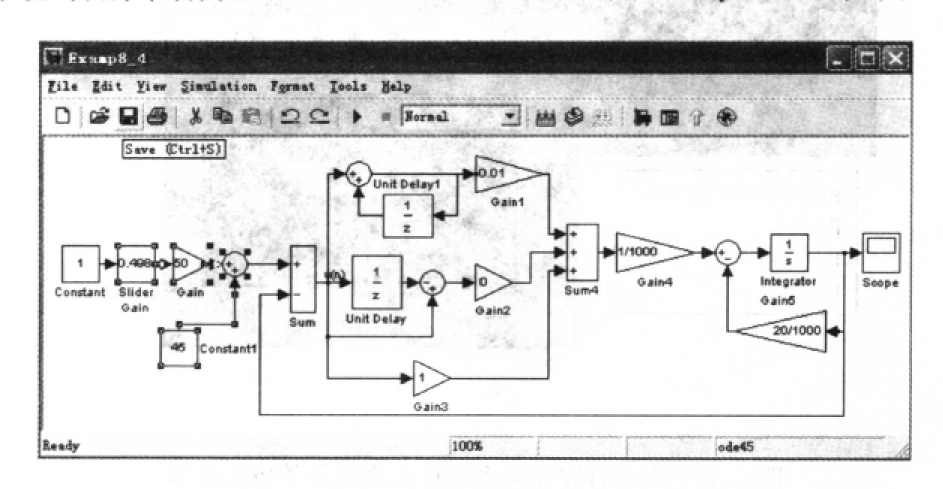


图 8.35 汽车行驶控制系统仿真模型

在创建的子系统中，Simulink 自动添加了子系统的输入模块 In1 和输出模块 Out1，用户可以从图 8.36 显示的打开的子系统中看到。

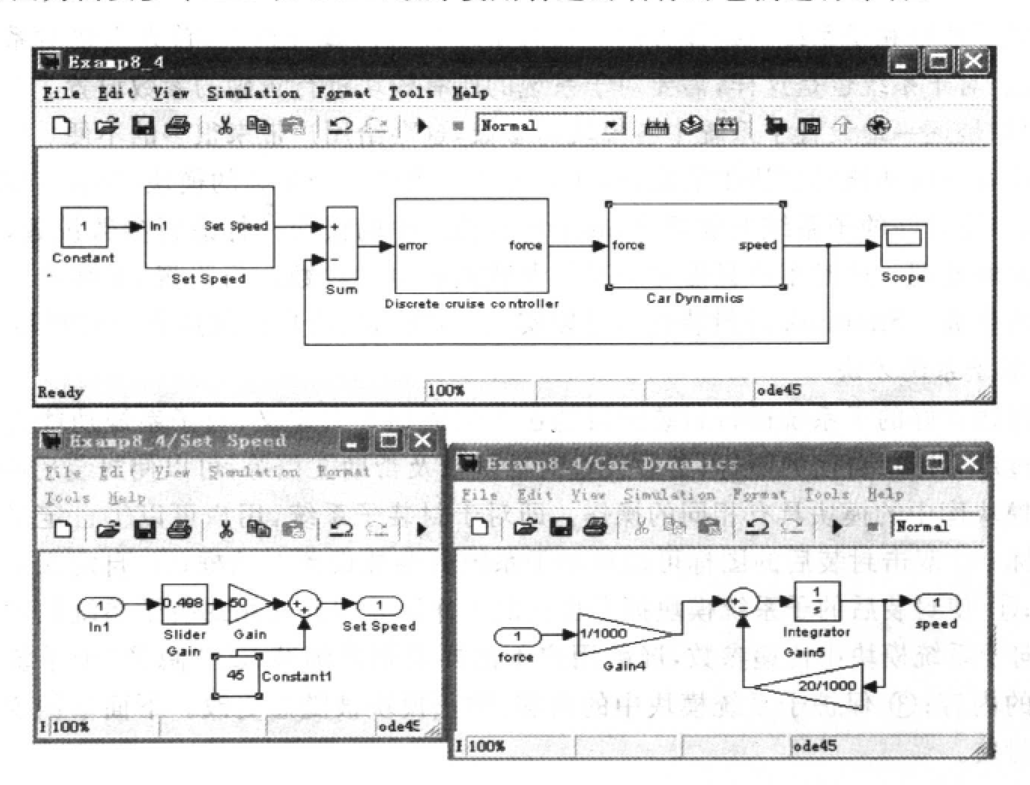
2. 自顶而下的设计

分析汽车行驶控制系统可知系统由 3 个功能模块组成：速度操纵机构的位置变换器、离散 PID 控制器、汽车动力机构。对于这种各部分功能较明确的系统，可以在建立系统模型时就考虑将各功能模块用不同的子系统实现。首先设计系统的总体模型，再进行细节设计。这种设计方法称作自顶而下的设计方法。

自顶而下的设计方法首先使用 Ports & Subsystems 模块库中的 Subsystem 建立子系统。这样建立的子系统内容是空的，然后双击此空子系统对其进行编辑。

需要说明的是，使用 Ports & Subsystems 模块库中的 In1 和 Out1 可以使子系统产生多

个输入端口和多个输出端口，这两种模块只是用来对信号进行传递，完成子系统和主系统之间的通信，不改变信号的任何属性。另外，信号标签可以越过它们进行传递。对于多输入多输出的子系统，因为需要多个 In1 和 Out1，最好使用合适的名称对它们进行命名。



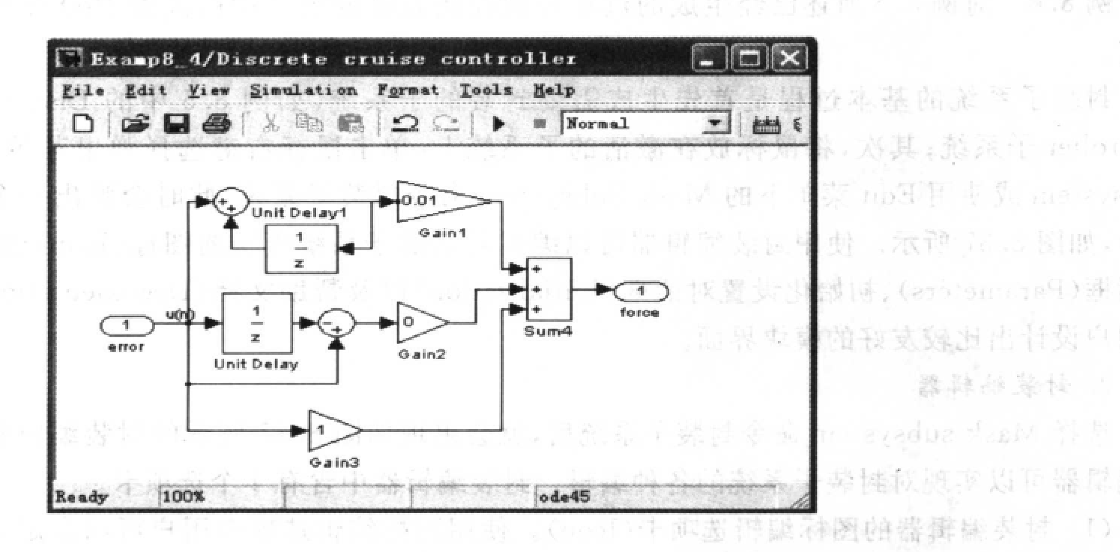


图 8.36　汽车行驶控制系统仿真模型及其各子系统

8.5.2　子系统的操作

在生成子系统之后，用户可以对子系统进行各种与系统模块相类似的操作，这时子系统相当于具有一定功能的系统模块。对子系统进行的操作可以是子系统的命名、子系统视图的修改、子系统的显示颜色等等，当然子系统也有其特有的操作，如子系统的显示、子系统的封装等等。

8.5.3 子系统的封装

使用子系统技术可以很好地改善系统模型的界面,使系统模型的可读性更好。前面介绍了如何生成子系统和子系统的操作方法。在对系统进行仿真分析时,首先需要对系统模块参数进行设置,对子系统也是这样,需要对子系统的所有模块进行正确的参数设置。在前面的介绍中,我们均是逐一地设置子系统中各模块的参数,这会给用户带来很多的不便。因为子系统一般均为具有一定功能的模块组的集合,在系统中相当于一个单独的模块,具有特定的输入输出关系,如果设计好的子系统能够像 Simulink 模块库中的模块一样进行参数设置,则会给用户带来很大的便利。这时用户只需对子系统参数选项中的参数进行设置,无须关心子系统的内部模块的组成。Simulink 的封装技术可以实现此功能,给用户提供这方面的便利。

1. 封装子系统方法

将已经建立好的子系统进行封装的目的在于生成用户自定义、与子系统功能完全相同的模块。通过定义用户自己的图标,参数设置对话框以及帮助文档等,可以使封装后的子系统与 Simulink 模块库中的模块具有相同的操作。即对于封装子系统,用户可以①自定义子系统模块及其图标;② 双击封装后的图标可以显示子系统的参数设置对话框;③ 自定义系统模块的帮助文档;④ 使封装后的子系统模块拥有自己的工作区。因此使用封装子系统技术具有以下优点:① 向子系统模块中传递参数,屏蔽用户不需要看到的细节;② “隐藏”子系统中不需要过多展现的内容;③ 保护子系统模块中的内容,防止模块被随意更改。下面举例说明如何进行子系统封装。

例 8.6 对例 8.5 所述已经生成的汽车行驶控制系统的模型中的离散 PID 控制器进行封装。

封装子系统的基本过程是首先生成需要封装的子系统,如例 8.5 中的 Discrete cruise controller 子系统;其次,将鼠标放在激活的子系统上,单击鼠标右键选择弹出菜单的 Mask Subsystem 或使用 Edit 菜单下的 Mask Subsystem 命令封装子系统,此时会弹出一个封装编辑器,如图 8.37 所示。使用封装编辑器可以编辑封装后子系统模块的图标(Icon)、参数设置对话框(Parameters)、初始化设置对话框(Initialization)以及帮助文档(Documentation),从而使用户设计出比较友好的模块界面。

2. 封装编辑器

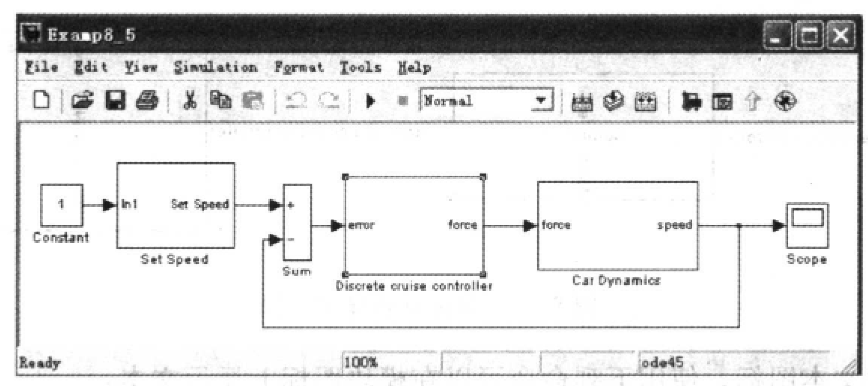
选择 Mask subsystem 命令封装子系统后,就会出现如图 8.37 所示的封装编辑器。使用此编辑器可以实现对封装子系统的各种编辑。封装编辑器中含有 4 个选项卡。

(1) 封装编辑器的图标编辑选项卡(Icon)。使用图标编辑选项卡用户可以自定义子系统模块的图标。虽然在默认状态下,封装子系统不使用图标,但友好的子系统图标可以使子系统的功能一目了然。

在图标编辑选项卡中,用户可以自定义子系统模块的图标。只需在图标编辑选项卡中的子系统模块绘制命令栏(Drawing commands)中使用绘图命令举例(Examples of drawing commands)中列出的绘图命令即可绘制模块图标,并可设置不同的参数控制图标界面的显示。下面逐一介绍各对话框的使用。

1)图标显示设置(Icon option)。图标边框设置(Frame):设置图标边框为可见(Visible)或不可见(Invisible);

图标透明性设置(Transparency):设置图标为透明(Transparency)或不透明(Opaque)显示。图标设置为透明(Transparency)时,图标后面的内容如模块端口标签可以显示出来。如图 8.38 所示,其中左侧为图标不透明设置结果,右侧为图标透明设置的结果。



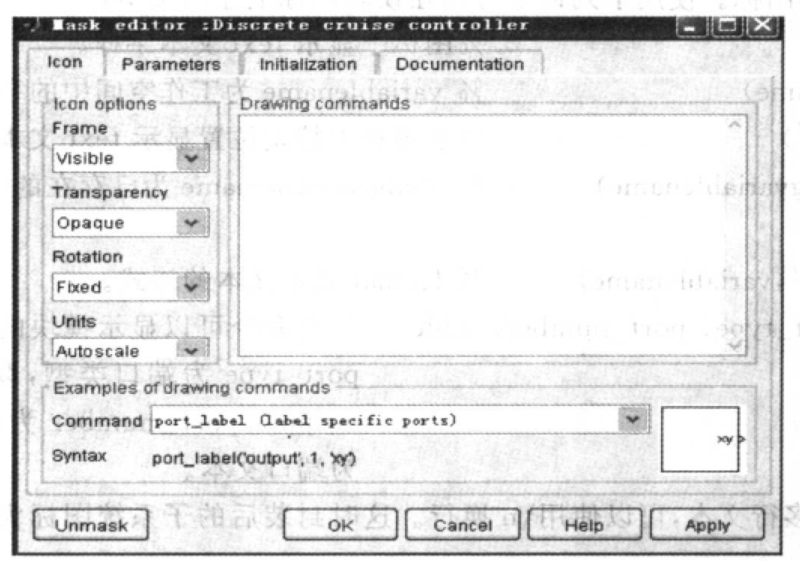


图 8.37　封装子系统及封装编辑器

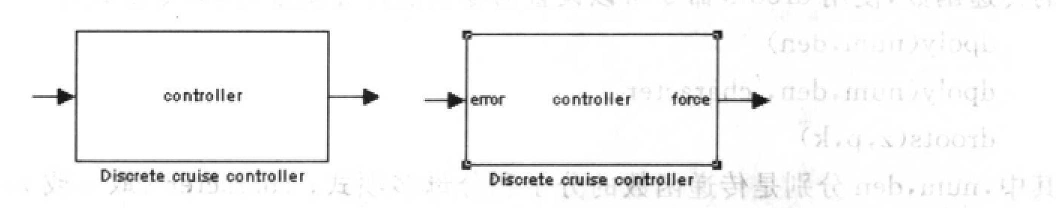


图 8.38　图标透明显示设置

图标旋转性设置(Rotation):设置图标为固定(Fixed)或可旋转(Rotates)。如图 8.39 所示,左侧为图标固定,图标"controller"不随模块的转动而转动;右侧为图标旋转,图标"controller"随模块的转动而转动。

图标单位设置(Units):设置图标绘制命令所使用的坐标系单位,仅对 plot 和 text 命令有效。其选项分别为自动缩放(Autoscale)、像素(Pixels)以及归一化表示(Normalized)。其中,Autoscale 表示图标自动适合模块大小,与其成比例缩放;Pixels 表示图标绘制采用像素作为单位;Normalized 表示模块大小为单位长度,绘制命令中的坐标值不得超过单位值 1。

2)图标绘制命令栏(Drawing commands)。封装后的子系统模块的图标均是在图标绘制

命令栏中完成绘制的。使用不同的绘制命令可以生成不同的图标。生成的图标可以是描述性文本、子系统数学模型图标、图像或图形等。如果在此栏中键入多个绘制命令，则图标的显示按照绘制命令的顺序显示。

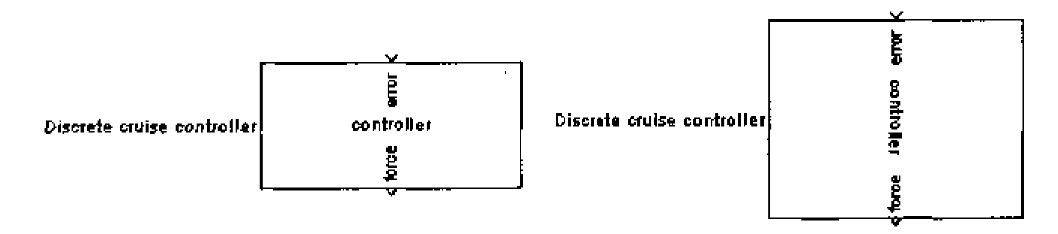


图 8.39 图标旋转显示设置

(i) 描述性文本图标。使用下列命令可以在模块图标上显示文本：

disp('text')	%图标上显示 text 文本字样。
disp(variablename)	% variablename 为工作空间中的字符串变量名。
text(x,y,'text')	%在图标上特定位置显示 text 文本字样。
text(x,y,stringvariablename)	% stringvariablename 为已存在的字符串变量名。
fprintf('text')	
fprintf('format',variablename)	% format 表示文本的格式。
port_label(port_type, port_number, lable)	%此命令可以显示模块的端口名称，其中 port_type 为端口类型，取值为'input'或'output'，port_number 为端口数目，label 为端口文本。

如果需要显示多行文本，可以使用\n 换行。这时封装后的子系统图标为描述性文本，如图 8.40(a)所示。

(ii) 子系统数学模型图标。使用 dpoly 命令可以将封装的子系统模块的图标设置为系统的传递函数，使用 droots 命令可以设置为零极点传递函数，其命令格式为

```
dpoly(num,den)
dpoly(num,den,'character')
droots(z,p,k)
```

其中，num,den 分别是传递函数的分子和分母多项式；'character'(取 s 或 z)为系统的频率变量；z,p,k 分别是传递函数的零点、极点和系统增益。需要注意的是，参数 num,den,z,p,k 必须是工作空间中已经存在的变量，否则绘制命令的执行将出现错误。例如，num=[1 2];den=[1 3 4 1];将子系统图标设置为子系统传递函数的命令如图 8.40(b)所示。

(iii) 图像或图形图标。使用 plot 或 image 命令可以将子系统模块的图标设置为图形或图像。其命令为

```
plot(x,y)
image(imread('photoname'))
```

例如，x=0:0.02*pi:2*pi; y=sin(x)；在图标命令栏键入 plot(x,y)可将子系统图标设置为 y-x 曲线，如图 8.41(a)所示。在图标命令栏键入 image(imread('helicopter.bmp'))，可将子系统图标设置为图像，如图 8.41(b)所示。

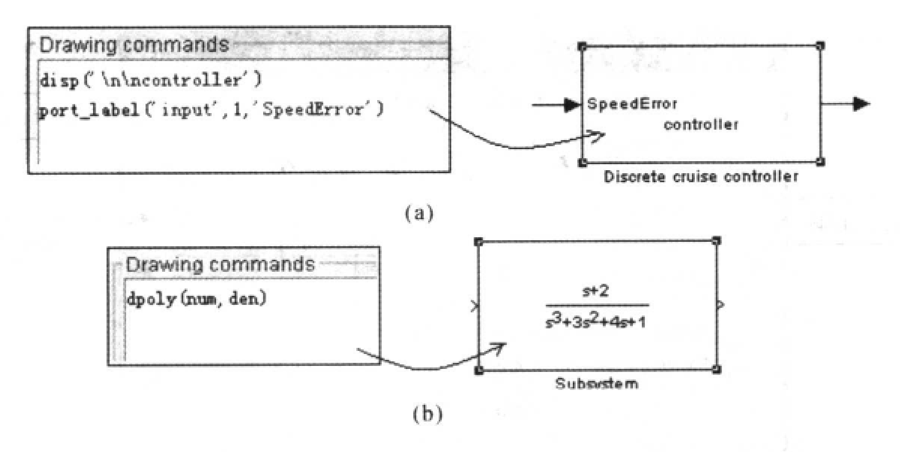


图 8.40　文本及传递函数图标绘制举例

(a) 文本图标绘制举例；(b) 子系统传递函数图标绘制举例

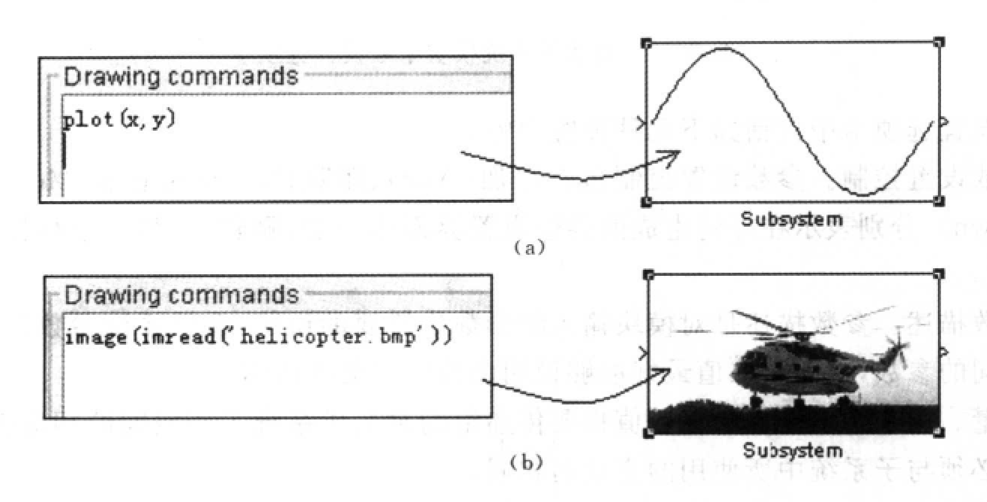


图 8.41　图形、图像图标设置举例

(a) 图形图标设置举例；(b) 图像图标设置举例

需要注意的是，在图标中绘制图像时，图像可以是 BMP 或 JPG 格式，如不是，需要使用图形处理工具箱中的 ind2rgb 命令进行转换。

(2) 封装编辑器的参数设置选项卡(Parameters)。子系统封装的目的之一是提供一个友好的参数设置界面。通常用户不需了解系统内部的细节，只需提供正确的模块参数即可完成对系统的设计与仿真。只有使用了子系统封装编辑器中提供的参数设置选项卡(Mask Editor 下的 Parameters 选项卡)进行子系统参数设置，才可以说是真正完成了子系统的封装，从而使用户设计出与 Simulink 模块库中的模块同样直观的参数设置界面。

不同于通常的子系统，封装的子系统具有独立的工作区。这是由于在没有对子系统进行封装之前，子系统中的模块可以直接使用 MATLAB 工作空间中的变量。通常的子系统可以看做是图形化的 MATLAB 脚本，即子系统只是将一些由模块实现的命令以图形化的方式组合而成的。而封装的子系统的内部参数对系统模型中的其他系统不可见，而且只能使用参数设置选项卡输入。

在例 8.6 中，对汽车行驶控制系统的模型中的离散 PID 控制器进行封装，并进行了图标绘制，这里仍针对该例说明如何对封装后的子系统的参数设置选项卡进行设置。在本封装子系统模块的参数设置界面中，应该提供控制律所需的 P, I 和 D 参数，如图 8.42 所示。

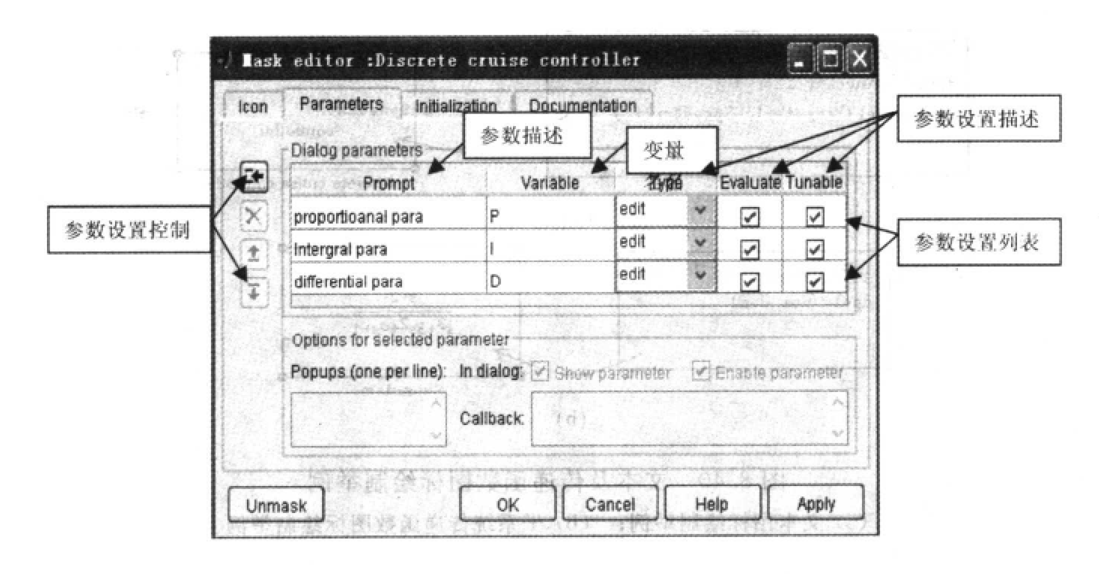


图 8.42　封装子系统模块系数设置选项卡

参数设置选项卡中包括如下几种设置内容：

1)参数设置控制。参数设置控制包括添加(Add)、删除(Delete)、上移(Move up)和下移(Move down),分别表示在即将生成的参数设置界面中添加、删除、上移与下移模块需要的输入参数。

2)参数描述。参数描述是对模块输入的参数作简要的说明,在子系统参数设置界面中用来区别不同的参数,因而其取值最好能够说明参数的意义或作用。

3)变量。用来指定键入的参数值将要传递给的封装子系统工作空间的相应变量,此处使用的变量必须与子系统中所使用的变量名相同。

4)参数设置描述。参数设置描述包括参数控制类型(Type)、是否为求值字符串(Evaluate)、是否可调整(Tunable)复选框。其中控制类型包括 Edit(需要用户在参数设置界面中键入参数值,适合多数情况)、Checkbox(复选框,表示逻辑值)和 Popup(在参数设置界面中弹出参数选项以便选择参数,弹出的参数选项值在 Popup 栏中输入)。

如图 8.42 所示,对封装子系统进行参数设置后,双击封装的子系统即可打开子系统的参数设置界面,如图 8.43 所示。用户只需在此界面中输入正确的参数即可进行系统的仿真设计分析。

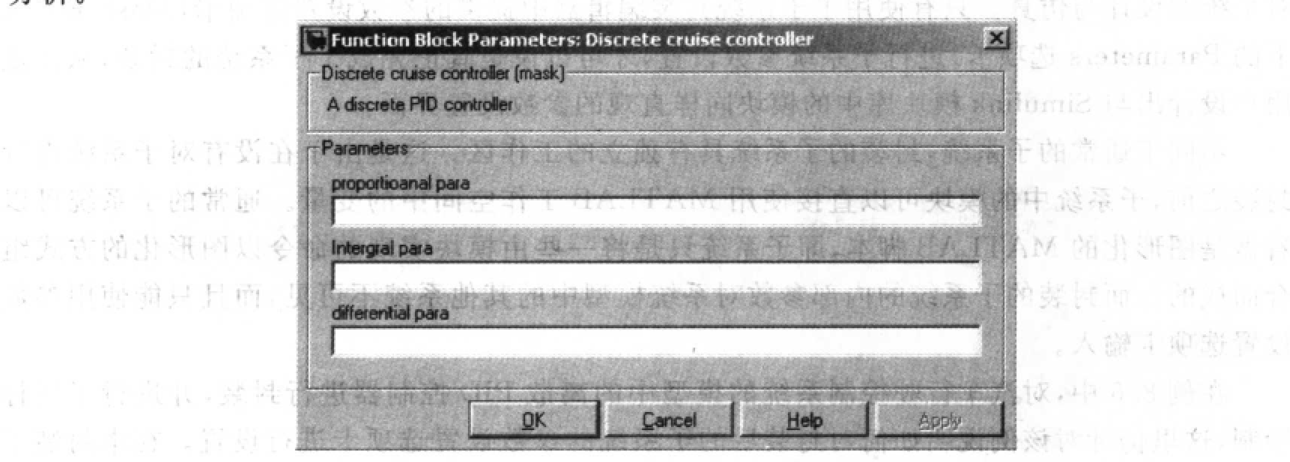


图 8.43　封装后子系统的参数设置界面

(3) 封装编辑器的初始化设置选项卡(Initialization)。初始化设置选项卡中的对话变量表是用户设置了参数设置选项卡后自动生成的。初始化命令一般为 MATLAB 命令，在初始化命令栏中可以定义封装后子系统工作空间中的各种变量，这些变量可以被封装子系统模块图标绘制命令、其他初始化命令或子系统中的模块使用。在下列情况下，Simulink 开始执行初始化命令：

模型文件被载入；

框图被更新或模块被旋转；

绘制封装子系统模块图标时。

(4) 封装编辑器的文档编辑选项卡(Documentation)。Simulink 模型库中的模块均提供了模块的简单描述和详细的帮助文档，方便用户的使用和理解。对于用户封装的子系统模块，封装编辑器的文档编辑选项卡可以使用户建立被封装子系统的所有帮助文档。对于例 8.6 封装的离散控制器的文档编辑如图 8.45 所示。

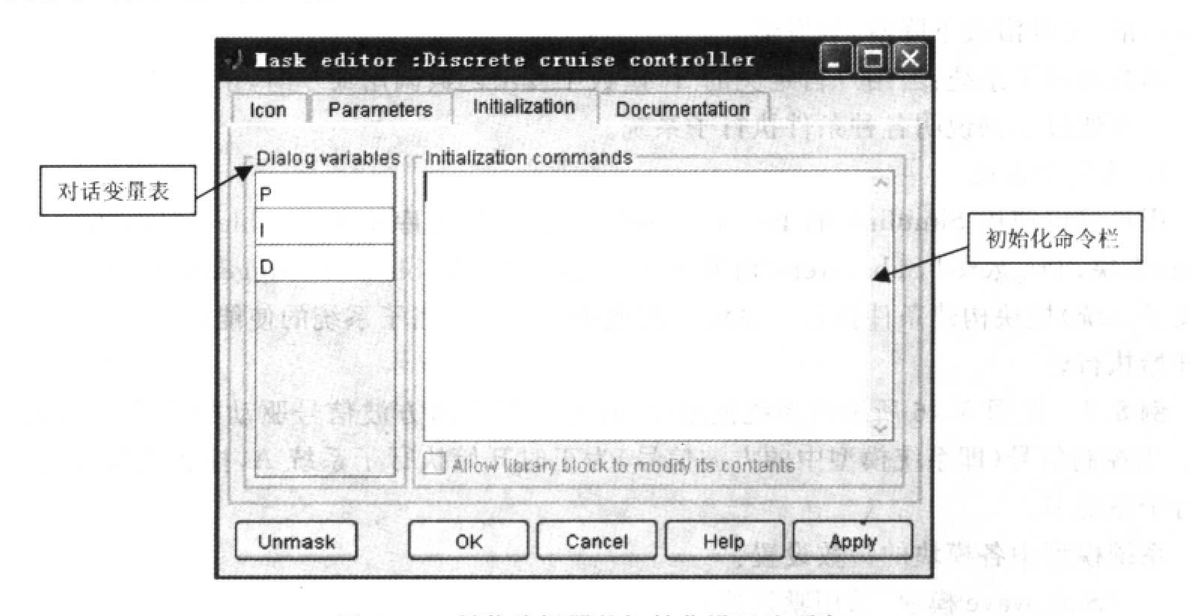


图 8.44　封装编辑器的初始化设置选项卡

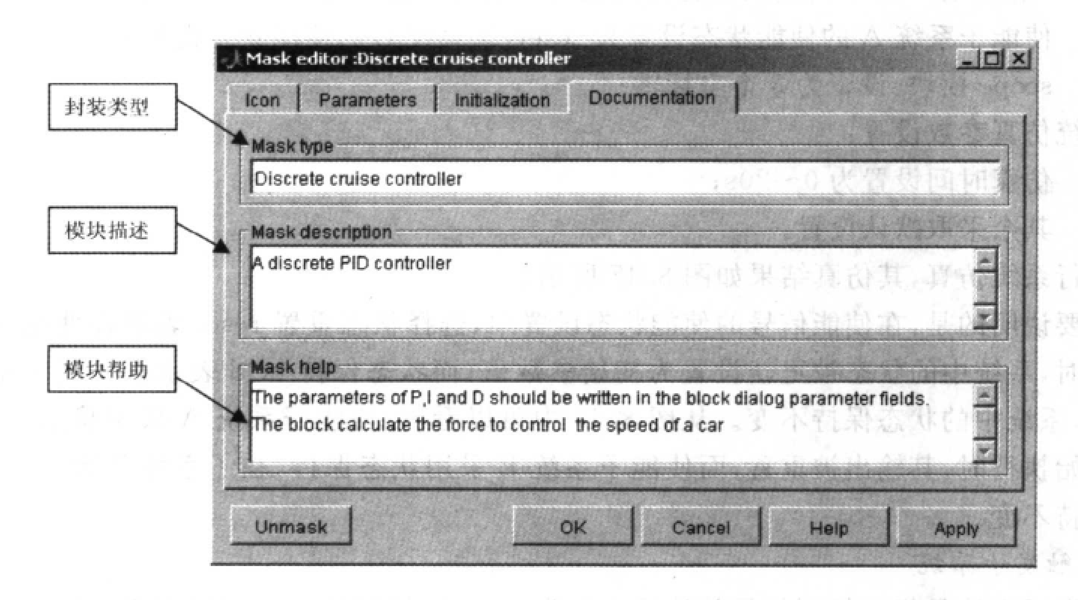


图 8.45　封装编辑器的文档编辑选项卡

8.5.4 Simulink 高级子系统技术

前面介绍的子系统的输出和输入具有一定的对应关系，对应一定的输入，子系统必定会产生输出。但在有些情况下，只有满足一定的条件时，子系统才被执行；即子系统的执行依赖于其他信号，这个称为控制信号的信号从子系统单独的端口即控制端口输入。这样的子系统称为条件执行子系统。在条件执行子系统中，子系统的输出不仅依赖于子系统本身的输入信号，而且受子系统控制信号的控制。

根据控制信号对条件执行子系统所控制方式的不同，可以将条件执行子系统划分为如下几种类型：

使能子系统：控制信号的值为正时，子系统开始执行；

触发子系统：当控制信号的符号发生变化时（即信号发生过零时），子系统开始执行。触发子系统的触发执行有三种形式：控制信号上升沿触发形式、控制信号下降沿触发形式和控制信号双边沿（上升沿或下降沿）触发形式。

函数调用子系统：当用户自定义的 S-函数中发出函数调用命令时，子系统开始执行。

下面通过示例说明各种条件执行子系统。

1. 使能子系统

用户可以使用 Simulink 的 Ports & Subsystems 模块库中的 Enable Subsystem（使能子系统）模块、Triggered Subsystem（触发子系统）模块及 Enable and Triggered Subsystem（使能触发子系统）模块构建条件执行子系统。使能子系统是指当子系统的使能信号为正时，子系统才开始执行。

例 8.7 在图 8.46 所示的系统模型中，存在着两个由方波信号驱动的使能子系统（A 和 B）。当控制信号（即系统模型中的方波信号）为正时开始执行子系统 A，控制信号为负时开始执行子系统 B。

系统模型中各模块的参数设置：

Sine wave 模块：采用默认值；

Signal generator 模块：设置为周期为 4 的方波信号，其余采用默认值；

使能子系统 A 的使能状态设置为 reset，子系统 B 的使能状态设置为 held。

scope 模块：设置为 3 个坐标轴。

系统仿真参数设置：

仿真时间设置为 0～20s；

其余采取默认设置。

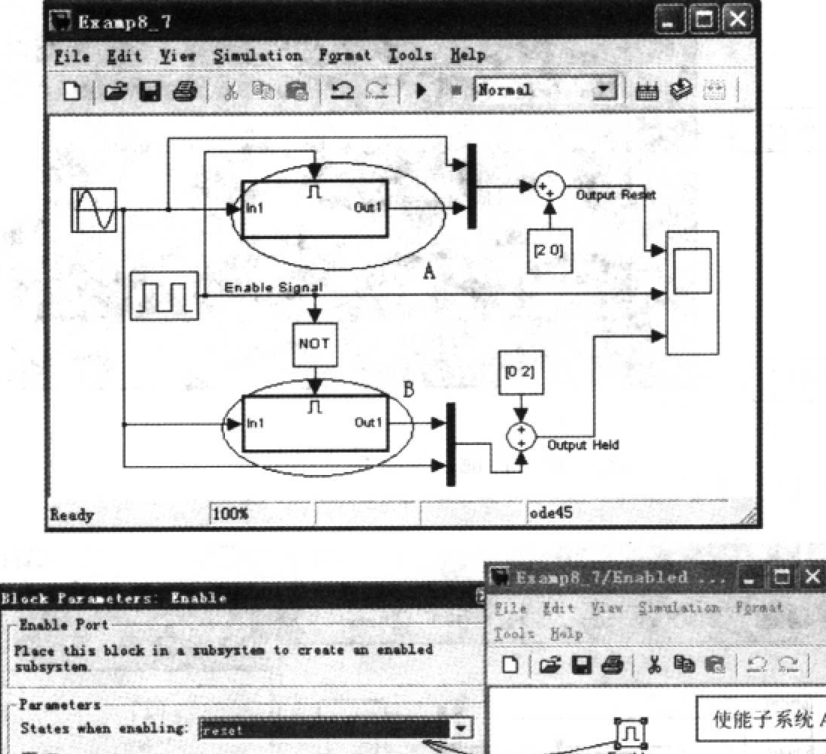
运行系统仿真，其仿真结果如图 8.47 所示。

需要说明的是，在使能信号的使能状态设置中，选择状态重置 reset 表示在使能子系统开始执行时，系统中的状态被重新设置为初始参数值；而状态保持 held 表示在使能子系统开始执行时，系统中的状态保持不变。从图 8.47 中可以看出，使能子系统 A 采用状态重置，在子系统开始执行时，其输出被重置；而使能子系统 B 采用状态保持，在子系统开始执行时，其输出被保持不变。

2. 触发子系统

触发子系统是指在控制信号的符号发生改变时（即控制信号出现过零事件时），子系统才

开始执行。根据控制信号符号改变方式的不同将触发子系统分为三类：上升沿触发子系统、下降沿触发子系统和双边沿（上升沿或下降沿）触发子系统。



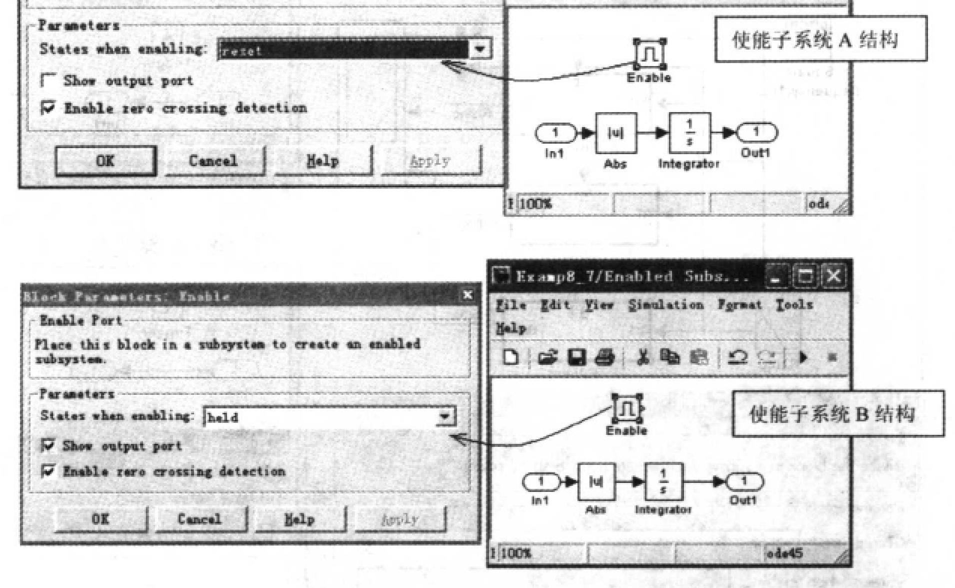


图 8.46　使能子系统模型及其使能参数设置

例 8.8　图 8.48 所示的系统模型中，有 3 个使用不同触发方式的触发子系统。

系统模型中各模块的参数设置：

Ramp 模块：斜率 slope 设置为 1，其余采用默认值；

Signal generator 模块：设置为周期为 1 的方波信号，其余采用默认值；

触发子系统的触发方式分别设置为 Rising，Falling 和 Either。

scope 模块：设置为 4 个坐标轴。

系统仿真参数设置：

仿真时间设置为 0～10s；

其余采取默认设置。

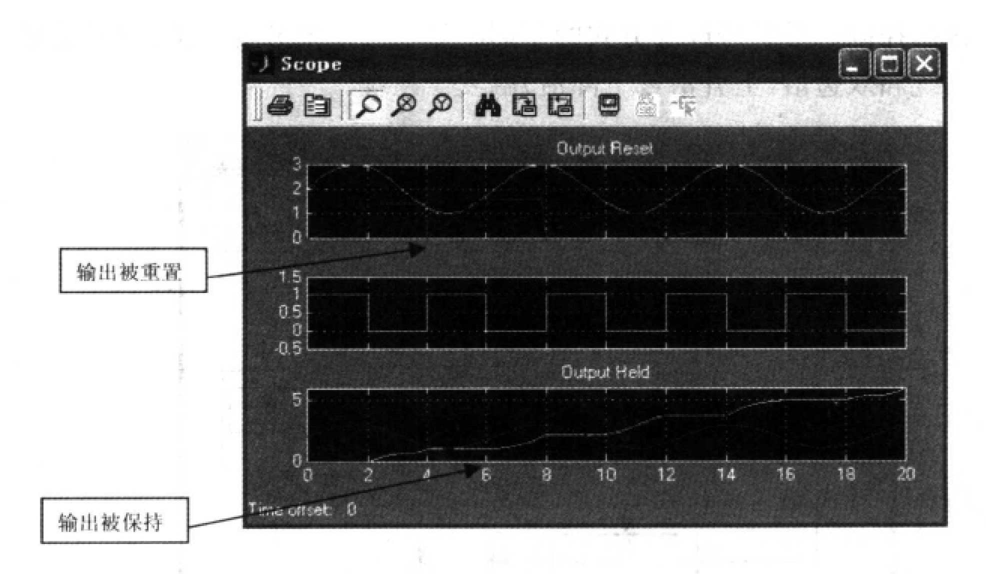


图 8.47　使能子系统仿真结果

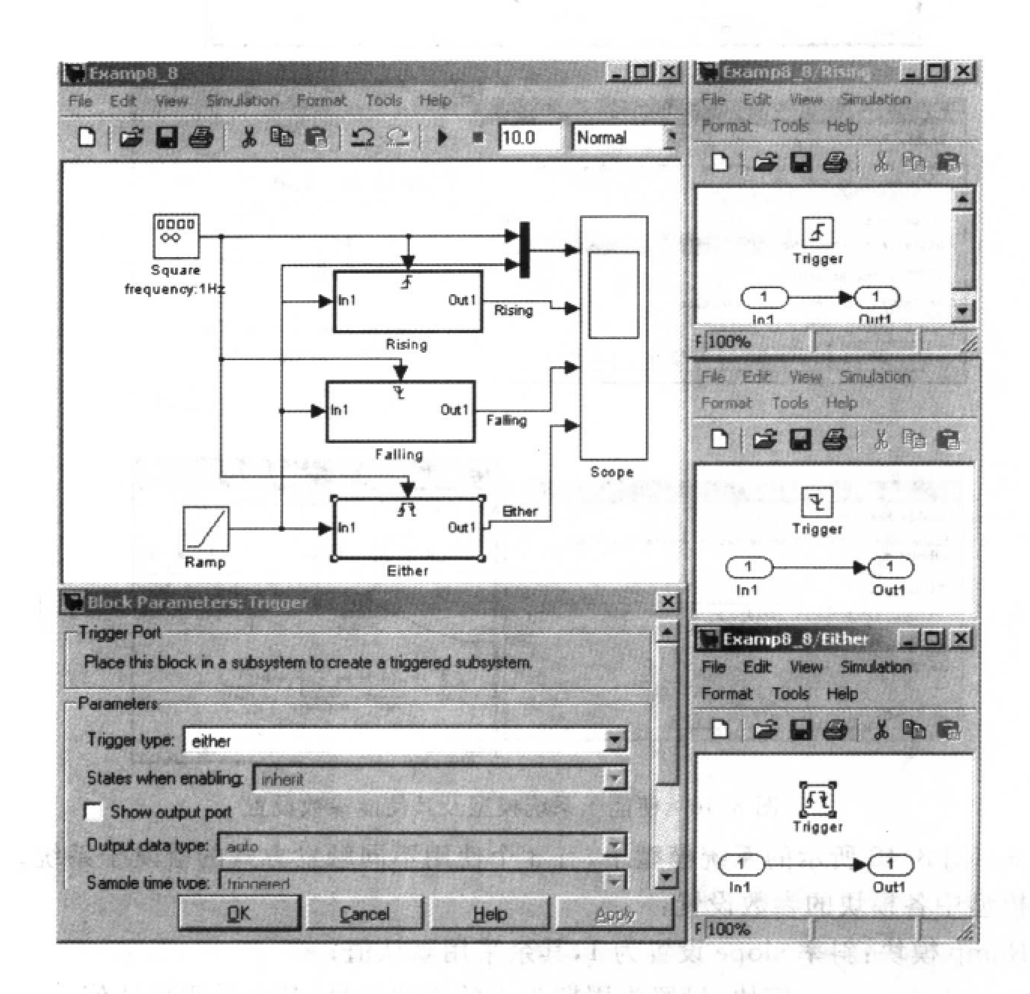


图 8.48　触发子系统模型及其参数设置

运行系统仿真，其仿真结果如图 8.49 所示。其中，第一个示波器输出系统正弦信号和触发控制方波信号。第二至第四示波器分别输出上升沿、下降沿和双边沿触发子系统的输出。

需要说明的是，触发子系统具有零阶保持特性，即系统在触发信号控制下开始执行的时刻，系统由输入产生相应的输出，当触发信号离开过零时，系统的输出保持在原来的输出值，不

发生变化。从仿真结果(见图 8.49)可以明显地看出触发子系统的零阶保持特性。此外,由于触发子系统的执行依赖于触发控制信号,因此对于触发子系统而言,不能指定常值的采样周期,只有带有继承采样时间的模块才能够在触发子系统中应用。

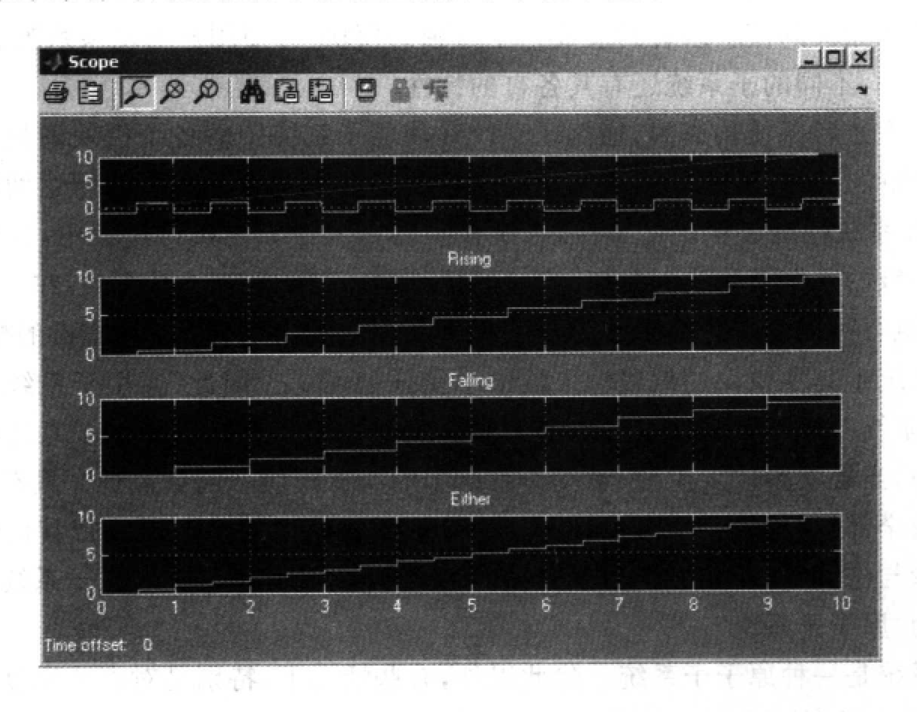


图 8.49　触发子系统模型仿真结果

3. 触发使能子系统

某些条件执行子系统的控制信号可能不止一个,很多情况下,条件执行子系统受到触发控制信号和使能控制信号的共同控制,只有当触发条件和使能条件同时满足时,子系统才开始执行,这样的条件执行子系统就是触发使能子系统。其工作原理如图 8.50 所示。系统等待一个触发事件的发生(即等待触发信号的产生),触发事件发生后,Simulink 检测使能信号,若使能控制信号为正,则子系统执行一次,否则不执行子系统。

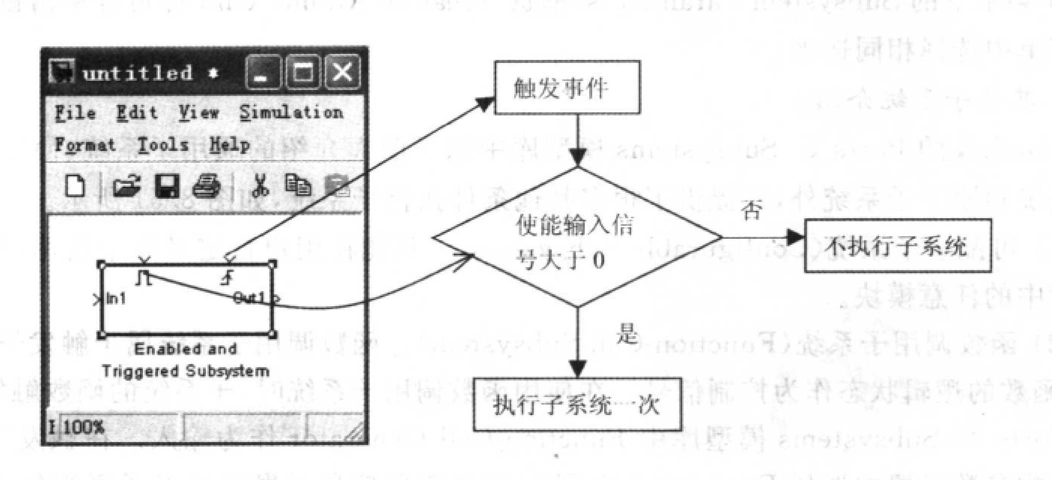


图 8.50　触发使能子系统工作原理

需要指出的是,条件执行子系统不允许同时使用多个 Trigger 信号或 Enable 信号,如果必须使用多个控制信号,用户可以先用逻辑操作符,将相关的控制信号先逻辑组合,以产生单

一的触发控制信号或使能控制信号。

4. 原子子系统

第 8.5 节介绍了通用子系统、使能子系统和触发子系统的概念和使用方法。虽然子系统都可以将系统中相关的模块组合封装成一个单独的模块，方便用户对复杂系统进行分析，但除了这些共同点外，不同的子系统还有其各自的特性。

Simulink 在进行系统仿真时，通用子系统和使能子系统中的各个模块的执行不受子系统的限制，即系统的执行与通用子系统和使能子系统的存在与否无关。使用这两种子系统均可以使 Simulink 系统模型更层次化，增强系统模型的可读性。在执行过程中，这两种子系统中的模块与上一级的系统模块统一排序，模块的执行顺序与子系统本身无关，在一个仿真时间步长中，系统的执行可以多次进出同一个子系统。两种子系统相当于一种虚设的模块组容器，其中各模块与系统中其他模块的信号输入输出不受任何影响。因此，通用子系统和使能子系统被称为“虚子系统”。然而，触发子系统的工作原理与通用子系统和使能子系统的均不同。在触发子系统中，当触发事件发生时，触发子系统中的所有模块一同被执行。只有在触发子系统中的所有模块全部执行完毕后，Simulink 才会转到系统模型的上一层执行其他模块。这种作为一个整体参与仿真，功能相当于一个单独的系统模块，且其中的模块在子系统中被排序执行的子系统称为“原子子系统”。

触发子系统是一种原子子系统。除此以外，有些情况下，特别是对多速率复杂系统作仿真分析时，需要将一个普通的子系统作为一个整体参与仿真计算。这是因为在多速率系统中，尤其是在生成系统可执行代码时，任何时序关系的差错都会导致整个系统仿真的失败，而且难以进行诊断分析。

在 Simulink 中建立原子子系统的方法有两种：

第一，使用 Ports & Subsystems 模型库中的 Atomic Subsystem 子系统模块建立一个空原子子系统，然后编辑该原子子系统；

其次，将已经建立的子系统强行转换成原子子系统。方法是先选择需要转换的子系统，选择 Edit 菜单下的 Subsystem Parameters，框选 Treat as Atomic Unit 即可或单击鼠标右键，在弹出菜单中选择相同选项。

5. 其他子系统介绍

Simulink 的 Ports & Subsystems 模型库中除了前面介绍的通用子系统、使能子系统、触发子系统和原子子系统外，还提供了很多其他条件执行子系统，如图 8.51 所示。

(1) 可配置子系统(Configurable Subsystem)。只能在用户自定义库中使用，代表用户自定义库中的任意模块。

(2) 函数调用子系统(Function-Call Subsystem)。函数调用子系统属于触发子系统，它使用 S-函数的逻辑状态作为控制信号。在使用函数调用子系统时，子系统的函数触发端口必须使用 Ports & Subsystems 模型库中 Function-Call Generator 作为输入。在触发子系统中触发信号的参数设置中选择 Function-Call 可以将由普通信号触发的触发子系统转换为函数调用子系统。

(3) For 循环子系统(For Iterator Subsystem)。For 循环子系统可以在一个仿真时间步长中循环执行子系统，用户可以指定在一个仿真时间步长中执行循环的次数。

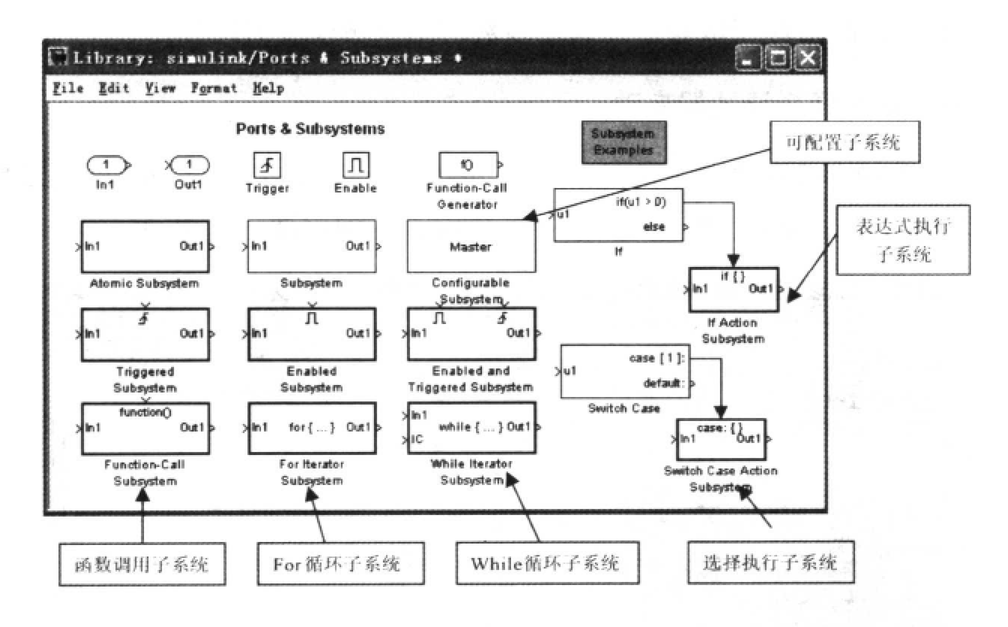


图 8.51　Port & Subsystems 模型库中的所有模块

(4) While 循环子系统(While Iterator Subsystem)。与 For 循环子系统类似，While 循环子系统同样可以在一个仿真时间步长内循环执行子系统，但是其执行必须满足一定的条件。While 循环子系统有两种类型：当型与直到型，这与其他高级语言中的 While 循环类似。

(5) 选择执行子系统(Switch Case Action Subsystem)。与 C 语言和 M 语言中的 Switch Case 语句的功能类似，在某些情况下，系统对于输入的不同取值分别执行不同的功能，选择执行子系统可以完成这种功能。需要指出的是，选择执行子系统必须同时使用 Switch Case 模块。

(6) 表达式执行子系统(If Action Subsystem)。与 C 语言和 M 语言中的 If-Else 语句的功能类似，表达式执行子系统的执行依赖于逻辑表达式的取值。表达式执行子系统必须同时使用 If 模块。

这里对 Ports & Subsystems 模型库中一些子系统模块的功能做了简单的说明，Ports & Subsystems 模型库中同时给出了这些子系统模块的使用算例，感兴趣的读者可以打开算例进行参考。

8.6　Simulink 的调试技术

功能强大、界面友好的调试功能是系统设计开发平台所必备的条件之一。Simulink 作为高性能的系统设计、仿真和分析平台，给用户提供了强大的模型调试工具。通过 Simulink 的调试工具，用户可以对动态系统模型进行调试，一种方法接着一种方法地运行仿真，以检查每种方法的仿真结果，发现其中可能存在的问题，并进行修改，从而快速完成系统设计、仿真和分析工作。

不同领域的不同系统模型的复杂程度相差悬殊，对系统模型进行调试的复杂程度也不相

同。Simulink 所提供的图形调试器可以满足多数应用领域模型的调试。

8.6.1 Simulink 图形调试器启动

使用 Simulink 的 Tools 菜单下的 Simulink debugger 命令或 Simulink 模型工具栏中的调试器按钮 可以启动调试器。图 8.52 所示是 Simulink 图形调试器窗口。

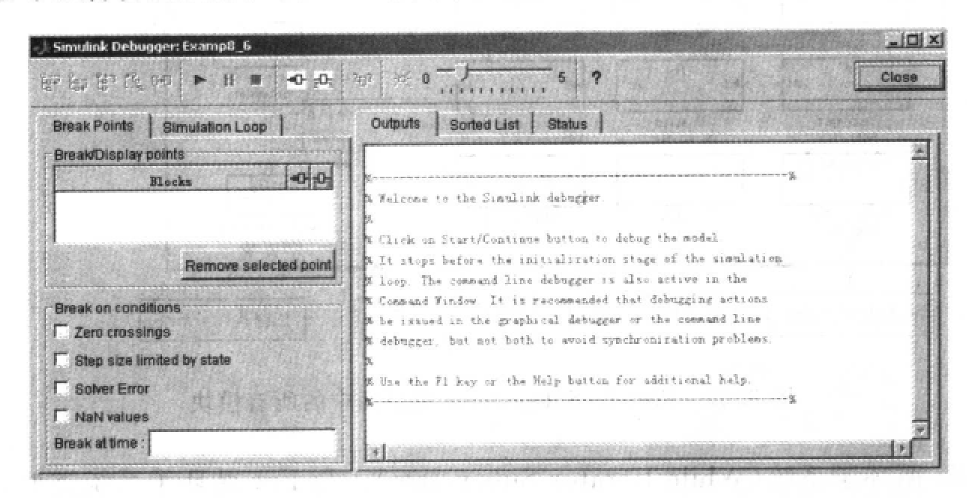


图 8.52 Simulink 图形调试器窗口

8.6.2 调试器的操作与功能

启动调试器，设置合适的调试断点之后，便可对系统模型中指定的模块和信号进行调试。Simulink 调试器能完成怎样的功能以及经过怎样的设置可以完成这样的功能？这些问题是用户在设置断点进行调试之前需要了解的。

1. Simulink 调试器工具栏

Simulink 调试器工具栏命令如图 8.53 所示。其工具栏包含的命令键的功能如表 8.1 所示。

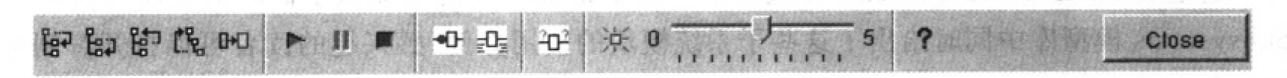


图 8.53 Simulink 调试器工具栏命令键

表 8.1 Simulink 调试器工具栏命令键使用说明

命 令 键	用 途
	进入下一种方法，中断于当前和下一种方法之间
	跳过下一种方法
	跳出当前状态
	在下一时间步中执行第一种状态

续 表

命令键	用　　途
	执行到下一模块
	开始或继续仿真
	暂停仿真
	中断仿真
	在指定模块之前设置断点
	执行时，显示指定模块的输入输出
	显示指定模块当前的输入输出
	选择动画方式开或关
?	调试器在线帮助
Close	关闭调试器

2. 断点显示及断点条件设置

Simulink 为用户提供了友好的调试界面。用户可以在断点显示框中了解到当前断点的信息，如断点位置、断点模块的输入输出等，如图 8.54(a)所示。很多情况下，用户除了用调试器在指定模块前设置断点外，还需要在一定条件下设置系统断点。Simulink 调试器提供了五种断点条件设置，如图 8.54(b)所示。

3. 仿真回路窗

选择 Simulation Loop 选项卡，即可打开仿真回路窗，如图 8.55 所示。Simulink 调试器的仿真回路窗包含 3 列：方法列、断点列和方法标识列。

(1) 方法列。方法列中显示仿真执行到当前所调用的各种方法，是按含可扩展、可压缩的结点给出的一种方法树。方法树上的每个结点表示需调用其他方法的方法。当仿真结束时，调试器显示仿真终止时执行的方法及直接或调用此方法的所有方法的名称。

(2) 断点列。断点列由一系列选项框组成。选择选项框意味着在左边所示的方法中设置一个断点。

(3) 方法标识列。方法标识列列出方法列中所列各方法的标识符。有些 Simulink 命令使用方法标识来指代方法的。方法标识是该方法在仿真过程中第一次激活时，Simulink 给其指定的一个整数标号。

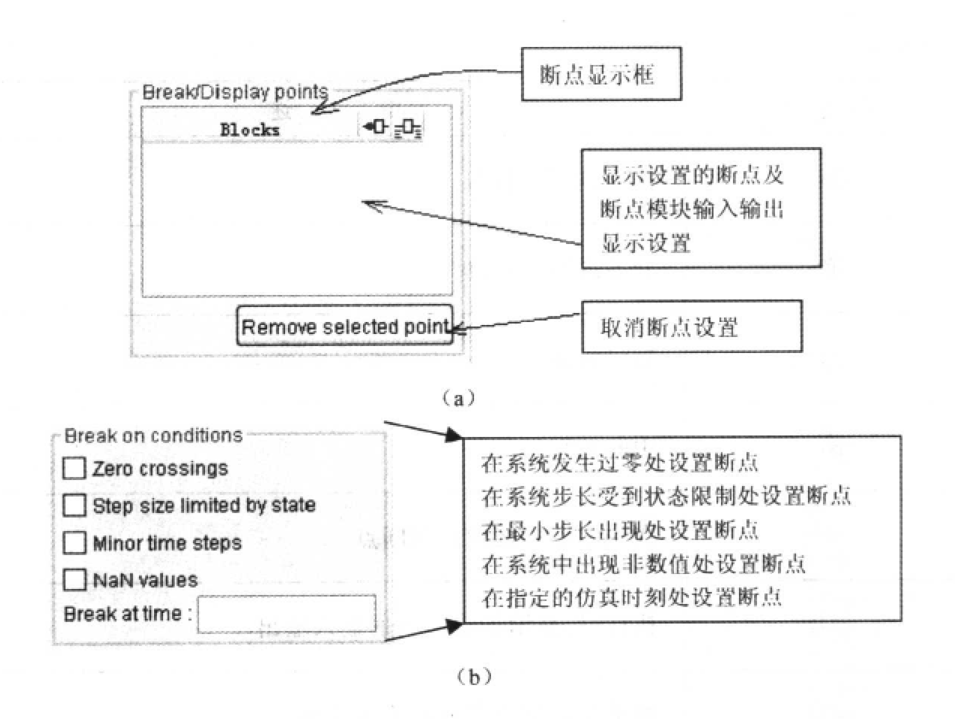


图 8.54 断点显示及断点条件设置

(a) 断点显示框； (b) 断点条件设置

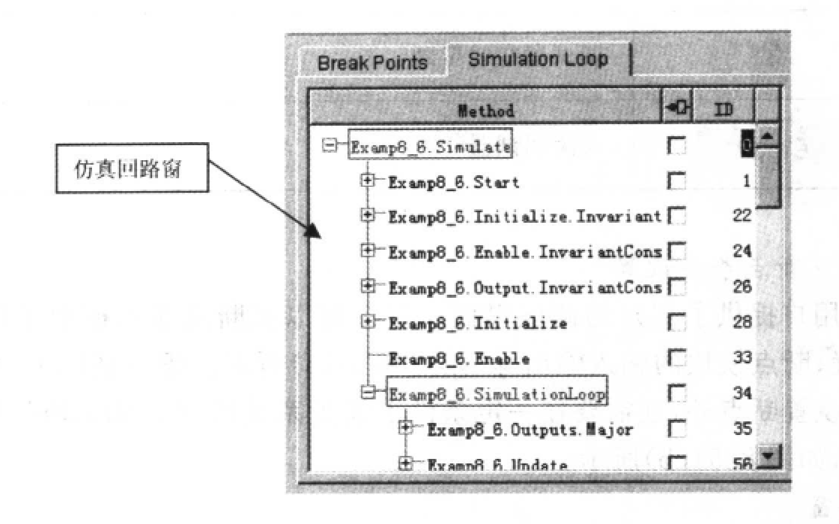


图 8.55 Simulink 调试器仿真回路窗

4. 调试器输出窗口

在对指定系统模型进行调试时，调试结果都可在 Simulink 调试器的输出窗口显示。图 8.56 所示是 Simulink 调试器输出窗口，输出调试结果，包括调试时刻、调试的模块及模块的输入和输出。

5. 调试器类型窗

选择调试器中的 Sorted list 选项卡，即可打开调试器类型窗。调试器类型窗显示模型根系统及其子系统模块类型及其执行顺序。

6. 状态窗

选择调试器中的 Status 选项卡，即可打开调试器状态窗。调试器状态窗显示调试器中各

种选项的值及其他一些状态信息。输出调试状态，包括当前仿真时间、缺省调试命令（执行至下一模块或执行至下一时刻）、调试断点设置及断点数等状态信息。

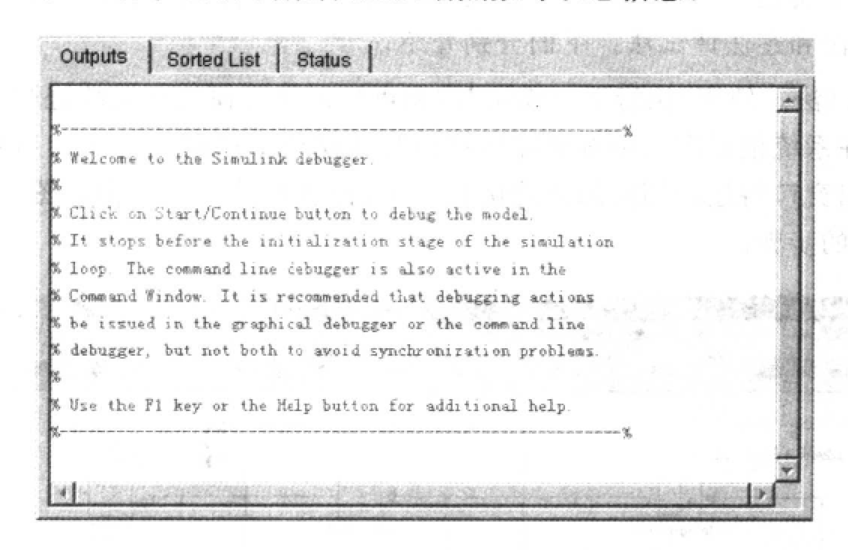


图 8.56　Simulink 调试器输出窗口

8.6.3　系统调试举例

下面仍以例 8.2 为例说明 Simulink 的调试技术，例 8.2 的系统仿真模型如图 8.28 所示。

1. 设置系统调试断点

用户必须先启动 Simulink 的调试器，才可设置断点。在积分模块（Intergrator 模块）之前设置断点：选择 Intergrator 模块，单击 Simulink 调试器工具栏中的"在指定模块之前设置断点"图标即可。此时断点显示框中将显示断点设置情况，如图 8.57 所示。

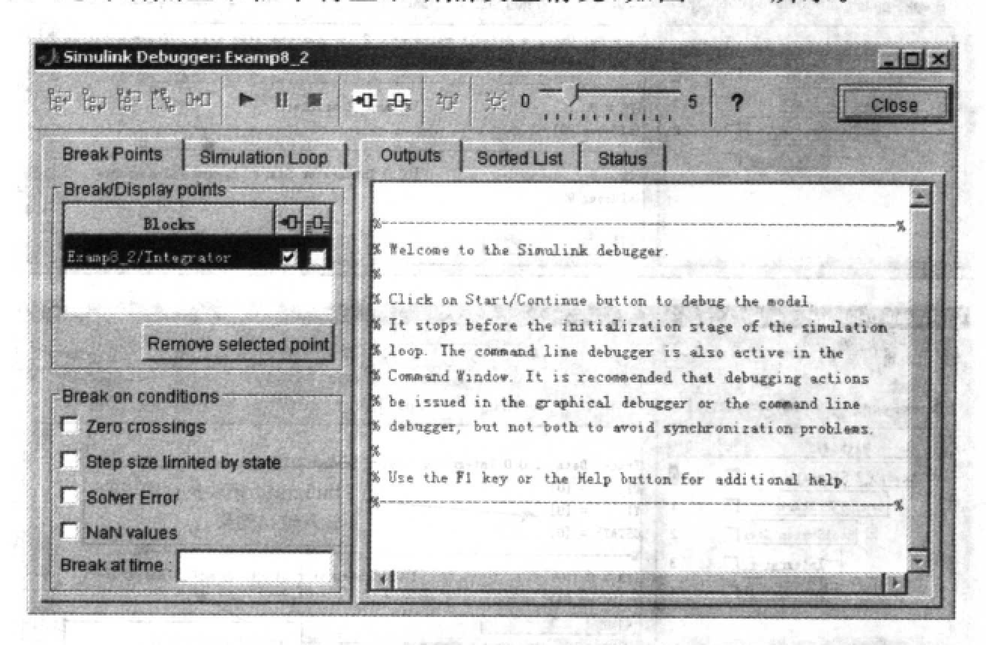


图 8.57　系统调试断点设置

如果需要，用户可以按照一定的断点设置条件设置系统调试断点。用户只需在相应的断点设置条件的复选框中选中即可。

2. 调试

点击调试器工具栏中的“开始执行”按钮▶，系统模型即进入调试模式。Simulink 调试的方法有逐块调试法和逐法调试法。下面分别介绍。

(1) 逐块调试法。使用“执行下一模块”按钮，即可逐模块对系统进行调试。在调试过程中，用户可以在系统模型窗口中看到即将被执行的模块会用箭头指示。当此模块执行完毕时，调试器的输出窗口将显示相应的系统仿真时刻、模块的输入与输出。图 8.58 显示了逐模块执行系统调试的步骤。

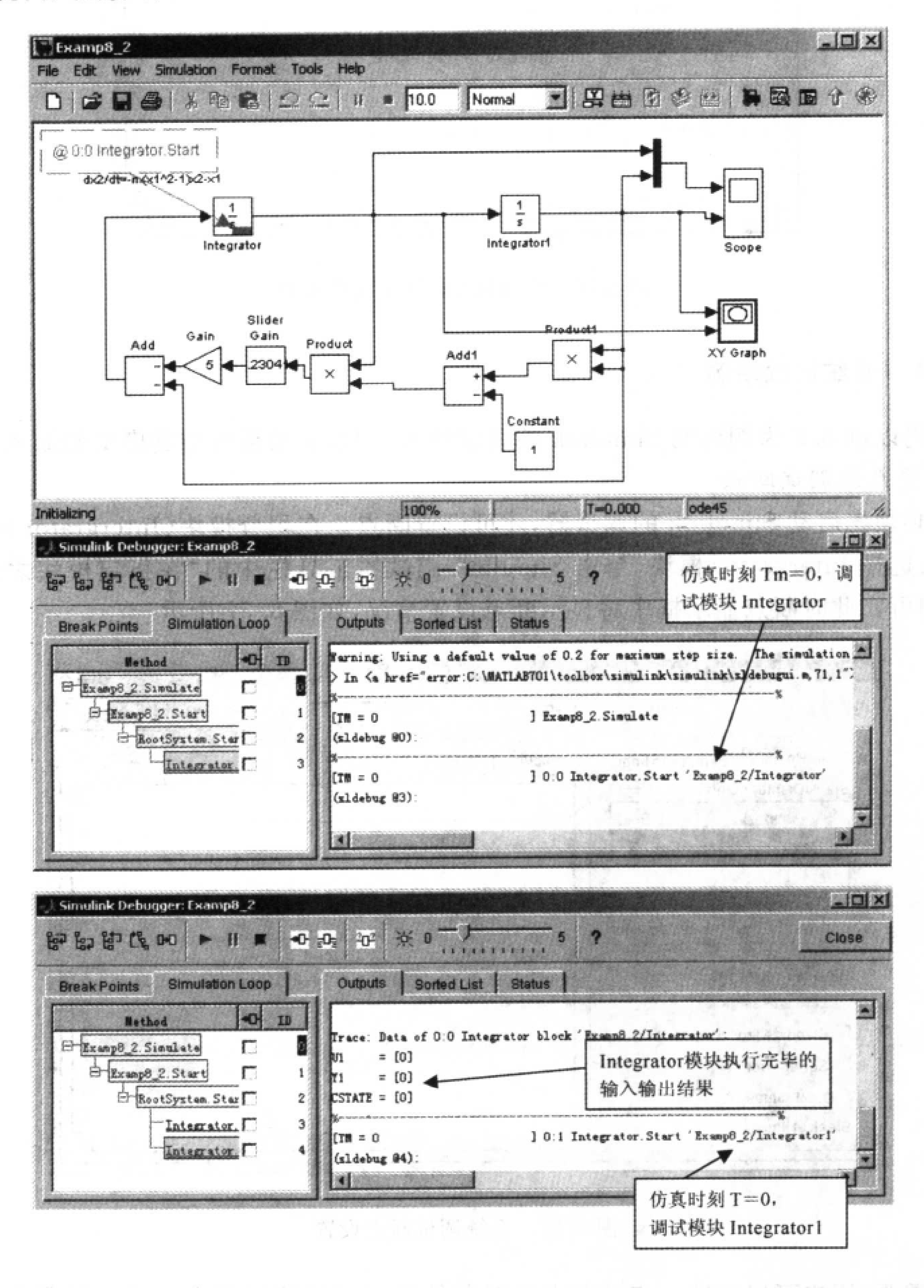


图 8.58　逐模块执行系统调试示意图

系统进入调试模式后，模块 Integrtor 首先被执行（在系统模型窗口中用箭头指示），此时在调试器输出窗口中显示系统仿真时刻 TM＝0，即将被执行的模块是 Integrator 模块；单击调试器工具栏“执行下一模块”按钮后，调试器执行 Integrator 模块，此时调试器的输出窗口显示 Integrator 模块的输入输出，将执行模块为 Integrator1 模块；如此逐模块地执行、逐模块地进行系统调试。

如果系统中含有子系统，则系统调试至子系统时，调试器会自动打开相应的子系统。子系统中的 Inport 模块（In1 模块）、Outport 模块（Out1 模块）和信号合并模块（Mux 模块）等“虚模块”均被调试器忽略。因为虚模块只是用来表示信号的某种操作，并不真正执行。

（2）逐法调试法。Simulink 调试器允许用户从当前执行的方法处逐个地运行仿真。使用 Simulink 调试左边的 4 个按钮，用户在每步调试时可以进入、跳过下一个方法或退出当前方法，还可以直接跳到仿真的开始。每步调试结束，调试器会显示仿真进行的方法以及进行到此法的结果。

点击调试器工具栏中的“开始执行”按钮▶，例 8.2 中的系统模型进入调试模式，点击一次 Simulink 调试器工具栏按钮或其他 3 个逐法调试按钮即做一步命令。在每步命令执行后，调试器在仿真回路窗中显示当前调用堆栈的方法，并用黄底色标记下一步要执行的方法，如图 8.59 所示。

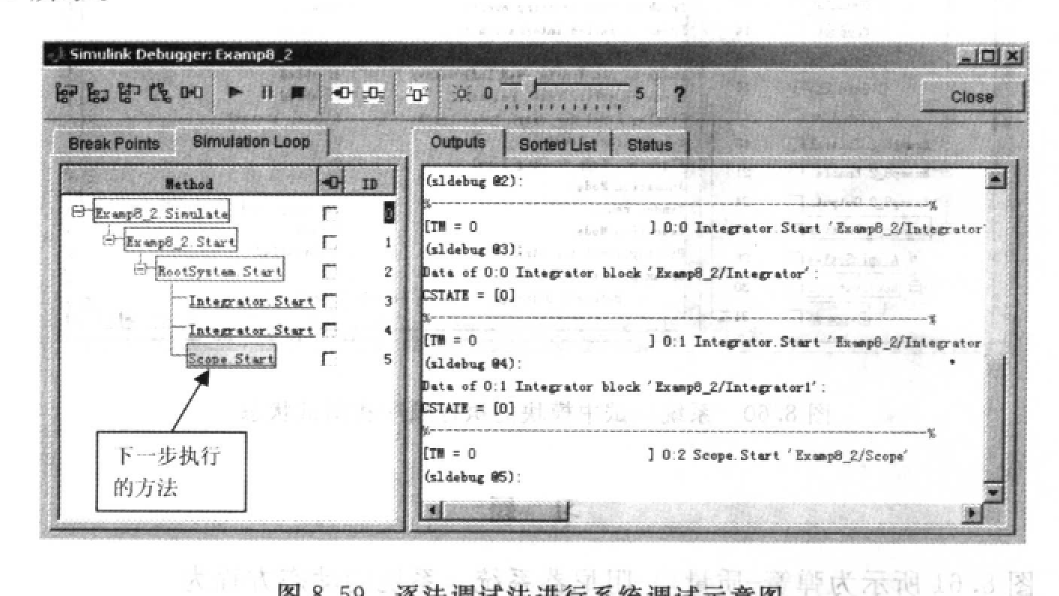


图 8.59　逐法调试法进行系统调试示意图

在系统调试过程中，Simulink 调试器按照一定的顺序对系统模块进行调试。使用调试器输出窗口的 Sorted List 窗口可以显示系统模块的执行顺序及其类型。此功能对于简单系统调试的作用不大，但对于大型复杂的多速率的动态系统来说是非常重要的。另外，如果系统中包含代数环，在 Execution Order 窗口中也会显示相关的系统模块。

此外，用户还可以在调试过程中随时对调试器所处的状态进行观察，此时需要使用调试器输出窗口中的 Status 窗口。图 8.60 所示是例 8.2 所述系统采用逐模块法调试时模块执行顺序和调试时的状态。

图 8.60 系统调试中模块的执行顺序和调试状态

习 题

8.1 图 8.61 所示为弹簧-质量块-阻尼器系统。系统的动态方程为

$$m\frac{\mathrm{d}^2x(t)}{\mathrm{d}t^2}+f\frac{\mathrm{d}x(t)}{\mathrm{d}t}+kx(t)=F(t)$$

式中，$x(t)$ 是质量块的位移，质量块质量 $m=5\ \mathrm{kg}$，阻尼器的阻尼系数 $f=0.5\ \mathrm{M\cdot s\cdot m^{-1}}$，弹簧的弹性系数 $k=5\ \mathrm{N\cdot m^{-1}}$；并且质量块的初始位移和初始速度均为 0。求当受到阶跃外力或扰动外力作用下，系统的响应（即质量块的位移 $x(t)$），并用示波器显示质量块位移随时间的变化。

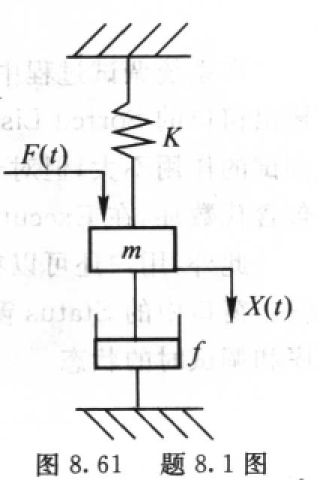


图 8.61 题 8.1 图

8.2 微分代数方程

$$
\begin{cases}
\dot{x} = -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\
\dot{x}_2 = 2x_1x_2 - 5x_2x_3 - 2x_2^2 \\
x_1 + x_2 + x_3 = 1
\end{cases}
$$

已知初始条件为 $x_1(0)=0.8, x_2(0)=0.1, x_3(0)=0.1$。构造 Simulink 模型，求 $x_1(t)$，$x_2(t)$ 和 $x_3(t)$。

8.3　典型 PID 控制系统方块图

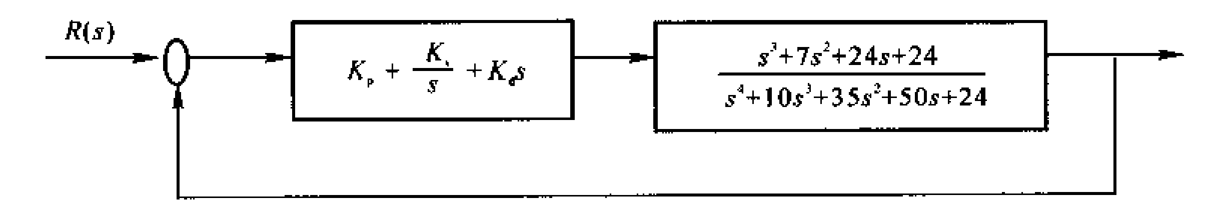


图 8.62　题 8.3 图

求当 $K_p=10$，$K_i=3$，$K_d=2$ 时，系统单位阶跃信号作用下的响应，并分析这些参数对系统性能的影响趋势。

8.4　在控制系统中，时常需要对信号计算其时域指标，如 ITAE 指标，定义为 $f(e)=\int_0^t \tau\,|e(\tau)|\,\mathrm{d}\tau$；ISE 指标，定义为 $f(e)=\int_0^t e^2(\tau)\mathrm{d}\tau$，其中，$e(\tau)$ 是系统的误差信号。建立 Simulink 模型，计算下列系统的 ITAE，ISE 指标。

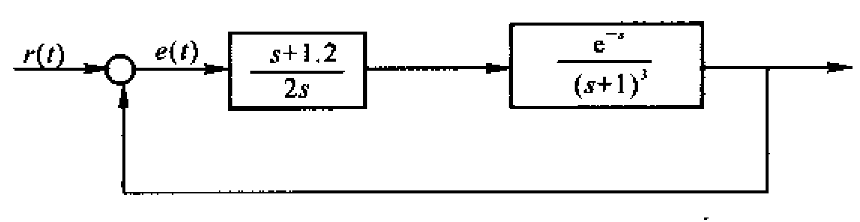


图 8.63　题 8.4 图

8.5　图 8.64 所示是简单的飞行控制系统，试建立此动态系统的 Simulink 模型，并进行简单的仿真分析。其中，$G(s)=\dfrac{25}{s(s+0.8)}$，系统输入为单位阶跃函数，$K_p=2$，$K_f=1$。

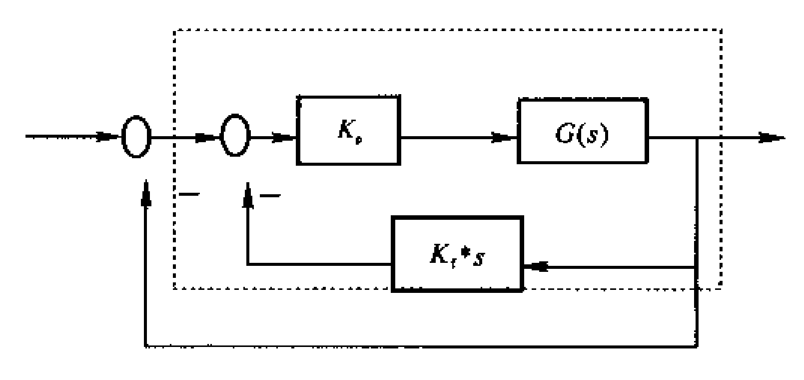


图 8.64　题 8.5 图

具体要求如下：

(1) 采用自顶向下的设计思路；

(2) 对虚线框中的控制器采用子系统技术；

(3) 用同一示波器显示输入和输出信号；

(4) 输出数据到 MATLAB 工作空间，并绘制图形。

8.6 在 MATLAB 命令窗口下键入 F14，即可打开 F14 飞机控制系统的 Simulink 仿真模块。阅读并理解此仿真的实现功能、求解器设置等信息。

8.7 将题 8.3 中 PID 控制器作为子系统封装起来，并在封装模块上标注文字。

第九章　Simulink 高级仿真技术

第八章对动态系统的建模、仿真与分析方法做了详细的介绍，这些方法足够用户对简单的动态系统进行仿真研究，但对于复杂的系统来说还略显不足。况且要想灵活高效地使用 Simulink，还必须了解 Simulink 的工作原理。本章主要介绍 Simulink 的高级仿真技术，包括 Scope 模块的高级使用技术、Simulink 的工作原理、过零事件、系统代数环的概念与解决方案、高级积分器的使用方法等。

9.1　Scope 模块的高级使用技术

从前面章节所举出的仿真示例中可以看出，在对系统进行仿真分析时，通常使用 Scope 示波器模块来观察动态系统的仿真结果或系统中指定的信号。用户可以很方便地对 Scope 模块进行各种设置以便对指定信号进行观测，对系统进行有效的分析。Scope 模块也可以设置成悬浮 Scope 模块，因而本节主要对 Scope 模块和悬浮 Scope 模块做详细的介绍。

9.1.1　Scope 模块的使用

Scope 模块是一个用途很广的显示模块，前面章节给出的 Simulink 仿真系统中多半都使用了这种模块，它是以图形的方式直接显示指定的信号。当无须对输出结果进行定量分析时，可以从 Scope 模块输出的曲线中直接获知系统的运动规律。Scope 模块给用户提供了很多设置方法，可以使用户对 Scope 模块的输出曲线进行各种控制调整，以便用户观测和分析输出结果。

Scope 模块的工具栏按钮命令如图 9.1 所示。下面分别介绍各项功能。

1. *打印输出*(Print)

将系统仿真结果的输出信号打印出来。

2. *视图自动缩放*(Autoscale)

点击此按钮可以自动调整显示范围以匹配系统仿真输出信号的动态范围。

3. *X 轴缩放、Y 轴缩放以及视图整体缩放*

此项功能可以分别对 X 坐标轴、Y 坐标轴或同时对 X，Y 坐标轴的信号显示进行缩放，以满足用户对信号做局部观察的需要。使用时，单击缩放按钮后选择需要观察的信号范围即可。若需要缩小视图，单击鼠标右键，选择弹出菜单的 Zoom out 即可。

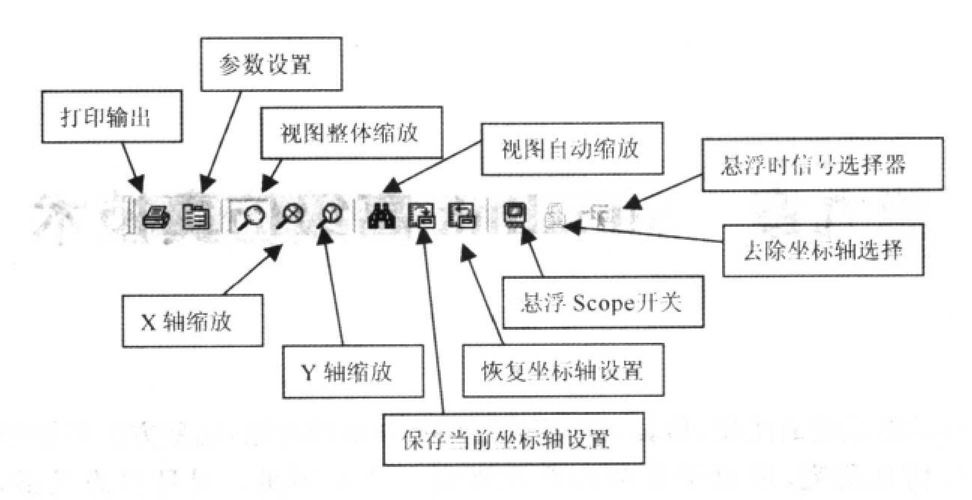


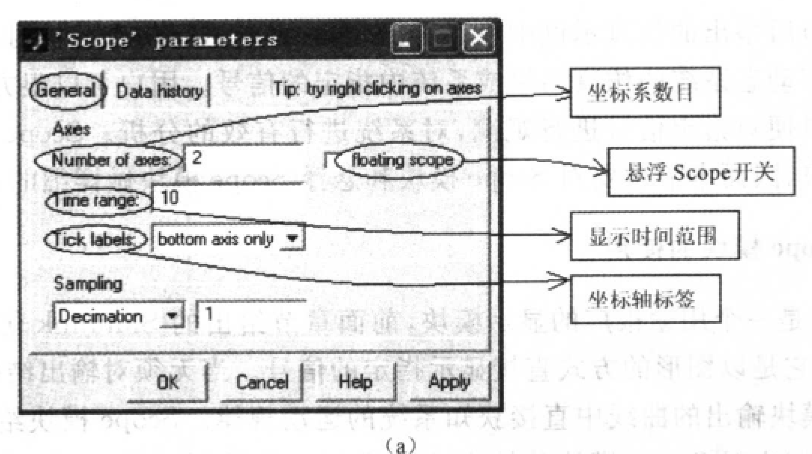
图 9.1　Scope 模块的工具栏命令键

4. 保存和恢复坐标轴设置

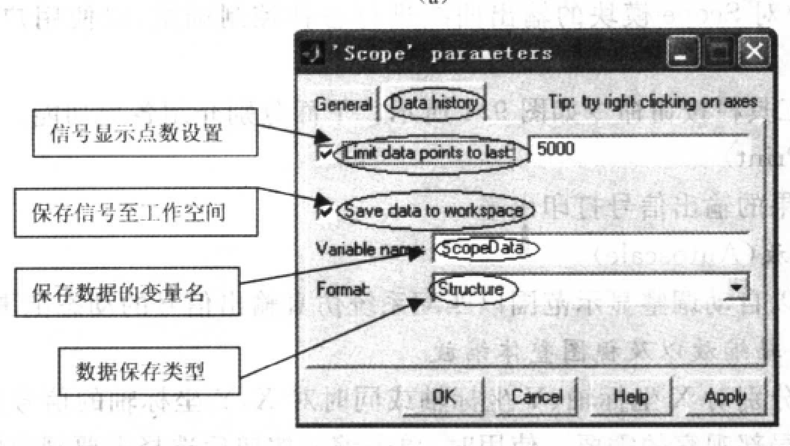
使用 Scope 模块观察输出信号时，用户可以保存坐标轴设置。这样，在信号的视图发生改变后，单击恢复坐标轴设置可以恢复以前保存的坐标轴设置。

5. Scope 参数设置

点击 Scope 模块工具栏的参数设置按钮(Parameters)，可以打开 Scope 模块的参数设置界面，如图 9.2(a)所示。Scope 模块的参数设置包含两个选项卡：General 和 Data History。



(a)



(b)

图 9.2　Scope 模块的参数设置界面

(a) Scope 模块的 General 选项卡；　(b) Scope 模块的 Data history 选项卡

(1) General 选项卡。通常参数设置界面首先显示 General 选项卡的内容。在 General 选项卡中可以进行下列设置：

1)坐标系数目(Number of axes)。在一个 Scope 模块中可以使用多个坐标系窗口同时输出多个信号。同时可使用的坐标系数目由此处设置。在默认设置下，Scope 模块仅显示一个坐标系窗口。

2)悬浮 Scope 开关(floating scope)。用来将 Scope 模块切换为悬浮 Scope 模块。悬浮 Scope 模块将在 9.1.2 中介绍。

3)显示时间范围(Time range)。用来设置信号的显示时间范围。需要注意的是信号显示的时间范围和系统仿真的时间范围可以不同。坐标系所显示的时间范围并非为绝对时间，而是指相对时间范围，坐标系左下角的时间偏移(Time offset)规定时间的起始时刻。

4)坐标系标签(Tick labels)。确定 Scope 模块中各坐标系是否带有坐标系标签。此选项提供 3 种选择：全部坐标系都使用坐标系标签(all)、最下方坐标系使用标签(bottom axis only)以及都不使用标签(none)。

(2) Data history 选项卡。在 Data history 选项卡中可以进行下列设置(见图 9.2(b))：

1)信号显示点数限制(Limit data points to last)。用来限制显示信号的数据点的数目，Scope 模块会自动对信号进行截取，只显示信号最后 n 个点(n 为设置的点数)。

2)保存信号至工作空间(Save data to workspace)。将 Scope 模块显示的信号保存至 MATLAB 工作空间中，以便于对信号进行更深入的定量分析。

3)数据保存变量名。设置被保存至 MATLAB 工作空间中数据的变量名。

4)数据保存类型。设置被保存至 MATLAB 工作空间中数据的保存类型。数据的保存类型有 3 种：带时间变量的结构体(Structure with time)、结构体(Structure)以及数组变量(Array)。

另外，在 Scope 模块中的坐标系中单击鼠标右键，选择弹出菜单中的坐标系属性(axes properties)，将弹出如图 9.3 所示的对话框。用户可以对 Scope 模块的坐标系标题和信号显示范围进行设置，以便于更好地分析显示信号。

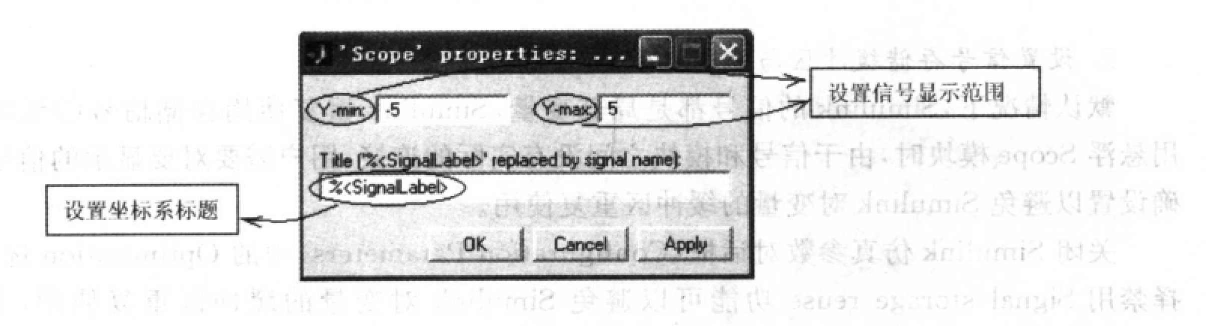


图 9.3　坐标系属性设置对话框

9.1.2　悬浮 Scope 模块的使用

在进行系统仿真分析时，用户往往需要对多个信号进行观察和做定性的分析。如果将每个信号都与一个 Scope 模块相连接，则系统模型中就会存在多个 Scope 模块，使得系统模型显得凌乱、不简练，且不易对不同 Scope 模块中显示的信号进行比较。使用悬浮 Scope 模块可以

解决这个问题。

与 Scope 模块不同，悬浮 Scope 模块没有输入端口，它在仿真过程中可以显示任何选定的信号，而 Scope 模块只能显示输入到其端口的信号。

这里以第 8.4 节中例 8.2 中所述连续的非线性系统的输出结果为例说明悬浮 Scope 模块的使用技术。图 8.28 求解 Van der Pol 方程的 Simulink 模型中 Scope1 是一个悬浮 Scope 模块。在 Simulink 模型中，悬浮 Scope 模块的创建有 3 种方法：第一，直接从 Sink 模型库中选择悬浮 Scope 模块；第二，点中普通的 Scope 模块的 parameters 中的 floating scope 选项，将普通的 Scope 模块设置为悬浮 Scope 模块；第三，点击图 9.1 所示的悬浮 Scope 开关也可将普通的 Scope 模块设置为悬浮 Scope 模块。

要使用悬浮 Scope 模块显示指定的信号，必须进行正确的设置。

1. 设置需要显示的信号

显示信号的选择是悬浮 Scope 使用的关键。使用悬浮 Scope 模块的信号选择器选择需要显示的信号。点击图 9.1 所示的悬浮时信号选择器即可打开信号选择器对话框，如图 9.4 所示，然后在可显示信号列表中选择需要显示的信号。

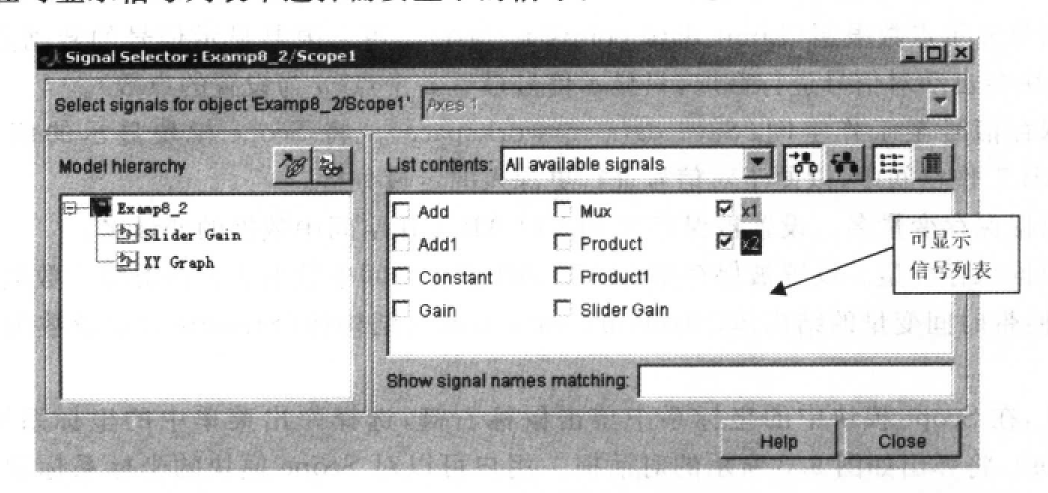


图 9.4　悬浮 Scope 模块的信号选择

2. 设置信号存储缓冲区与全局变量

默认情况下，Simulink 的信号都是局部变量，Simulink 重复使用存储信号的缓冲区。使用悬浮 Scope 模块时，由于信号和模块之间没有实际的连接，用户需要对要显示的信号进行正确设置以避免 Simulink 对变量的缓冲区重复使用。

关闭 Simulink 仿真参数对话框（Configuration Parameters）中的 Optimization 选项卡，选择禁用 Signal storage reuse 功能可以避免 Simulink 对变量的缓冲区重复使用，如图 9.5 所示。

对于例 8.2，使用悬浮 Scope 模块的信号选择器选择需要显示的信号 x_1 和 x_2，在进行了必要的信号存储缓冲区及全局变量设置后，重新运行，悬浮 Scope 模块的显示的仿真结果如图 9.6 所示。

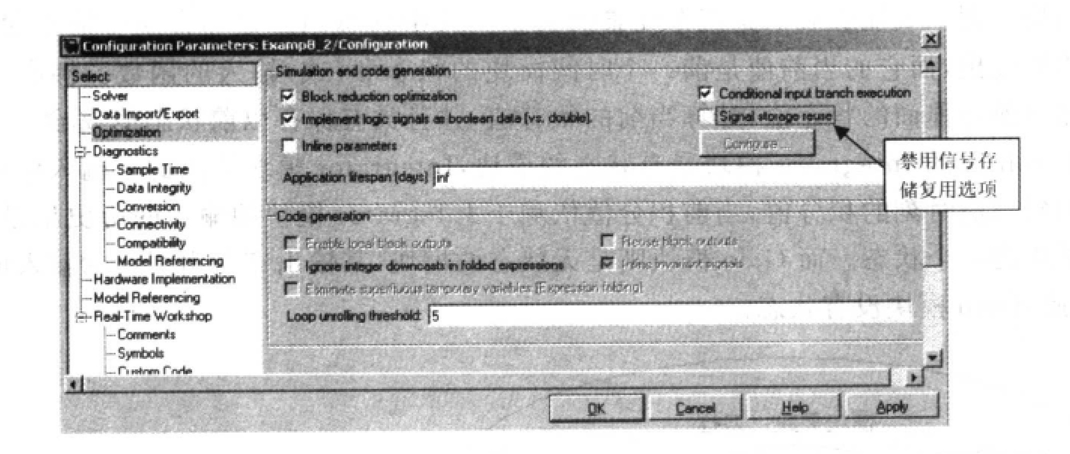


图 9.5 信号存储缓冲区设置

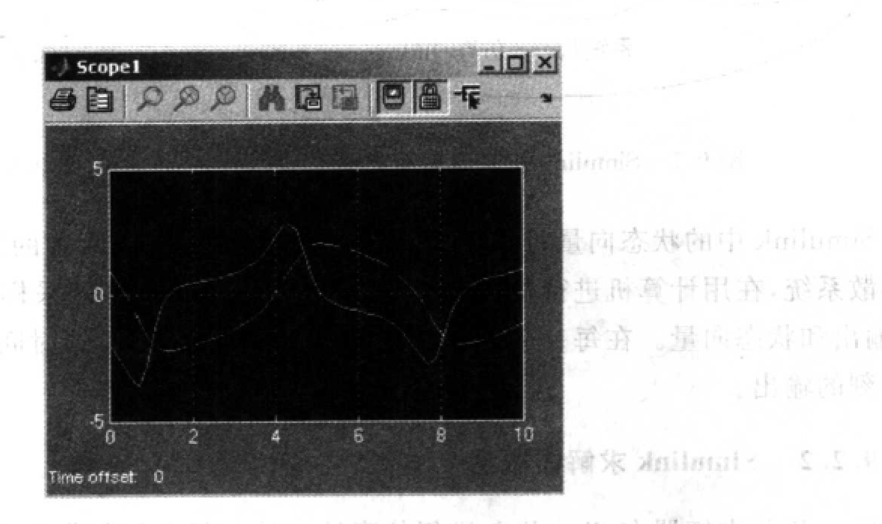


图 9.6 悬浮 Scope 显示的例 8.2 的仿真结果

9.2 Simulink 的工作原理

尽管 Simulink 用户提供了一个具有友好用户界面的系统级仿真平台，通过它的图形化仿真环境，为用户屏蔽掉了许多繁琐的编程工作，而把主要精力放在模型的构建上，从而使用户能够快速完成系统的设计任务。但为了能够高效灵活地使用 Simulink，必须了解 Simulink 的工作原理。Simulink 是通过系统模型（框图）与 MATLAB 求解器直接的交互对话完成系统仿真的，如图 9.7 所示。Simulink 传递模块参数和差分（微分）方程给 MATLAB 求解器，而 MATLAB 求解器计算系统模块的输出以更新离散系统的状态并确定下一步仿真时间。

9.2.1 系统模型

简单地说，Simulink 中的每个模块都是一个具有输入、输出和状态三个基本元素的系统。在 Simulink 中，模块都是用向量来表示这三个基本元素的，假设 u，x 和 y 分别表示输入、状态

和输出向量。图 9.8 能够表示这三个元素的关系。其中状态向量是非常重要的概念，状态决定了模块的输出，而它的当前值是前一个时间模块的状态和(或)输入的函数。拥有状态的模块必须能够保存前面的状态值，计算当前的状态值，并且具有保存以前状态值或输入值的存储空间。Simulink 的 Integrator 模块是有状态的模块，Integrator 模块输出的是输入信号从仿真开始时刻到当前时刻的积分值，当前积分值依赖于 Integrator 模块的输入的历史记录，因此积分值是模块的一个状态。而 Gain 模块则是无状态的模块，其输出完全由当前的输入值和增益决定，因此，Gain 模块没有状态。

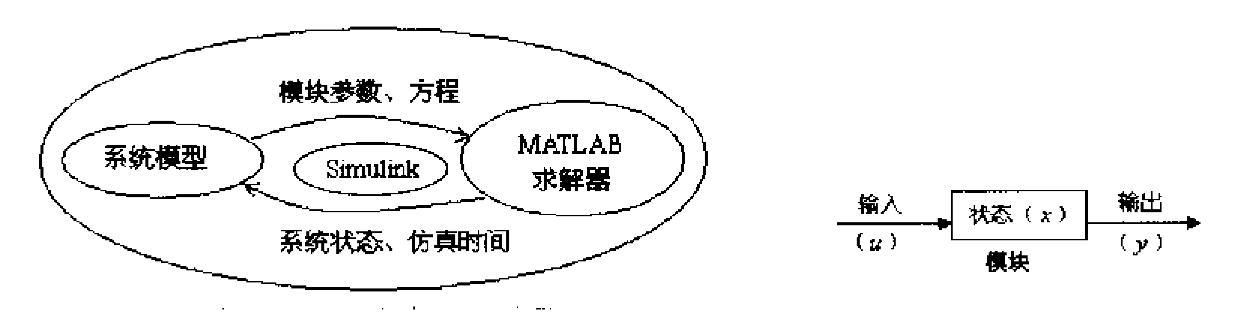


图 9.7 Simulink 仿真原理示意图

图 9.8 Simulink 模块的基本模型

Simulink 中的状态向量可以分为连续状态、离散状态或两者的结合。无论是连续系统还是离散系统，在用计算机进行仿真时，都需要在采样时间点(即采样时间步长)估计系统的输入、输出和状态向量。在每一个采样时刻，Simulink 根据当前的时间、输入和状态来决定该采样时刻的输出。

9.2.2 Simulink 求解器概念

Simulink 求解器在 Simulink 进行仿真计算的过程中起着非常重要的作用，它是 Simulink 进行仿真计算的核心。因此，要了解 Simulink 的工作原理，必须先对 Simulink 求解器有所了解。

1. 离散求解器

离散系统一般是用差分方程描述的，其输入与输出仅在离散的采样时刻取值，系统的状态每隔固定的时间才更新一次，而 Simulink 对离散系统的仿真核心是对离散系统差分方程的求解。因此，Simulink 可以做到对离散系统仿真的绝对精确(除了有限的数据截断误差)。

要对纯粹的离散系统进行仿真，需要使用离散求解器对其进行求解。用户需要选择 Simulink仿真参数设置对话框中的求解器选项卡中的 discrete(no continuous states)选项，即没有连续状态的离散求解器，便可对离散系统进行精确的求解与仿真(见例 8.3)。

2. 连续求解器

与离散系统不同，连续系统的输入、输出与状态都是连续的，并且输入、输出与状态的关系需要用微分方程描述。因此需要使用数字计算机对系统的微分或偏微分方程进行求解，所以只能求出其数值解(即近似解)，不可能得到系统的精确解。

Simulink 对连续系统进行仿真，实质上是对系统的微分或偏微分方程进行求解。对微分方程的近似求解的方法有多种，因此 Simulink 的连续求解器有多种不同的形式，如变步长求解器 ode45，ode23 和 ode113，定步长求解器 ode5，ode4 和 ode3 等。采用不同的连续求解器会

对连续系统的仿真结果和仿真速度产生不同的影响，但一般不会对系统的性能分析产生较大的影响，因为用户可以设置具有一定误差范围的连续求解器进行相应的控制。连续求解器设置如图 9.9 所示。

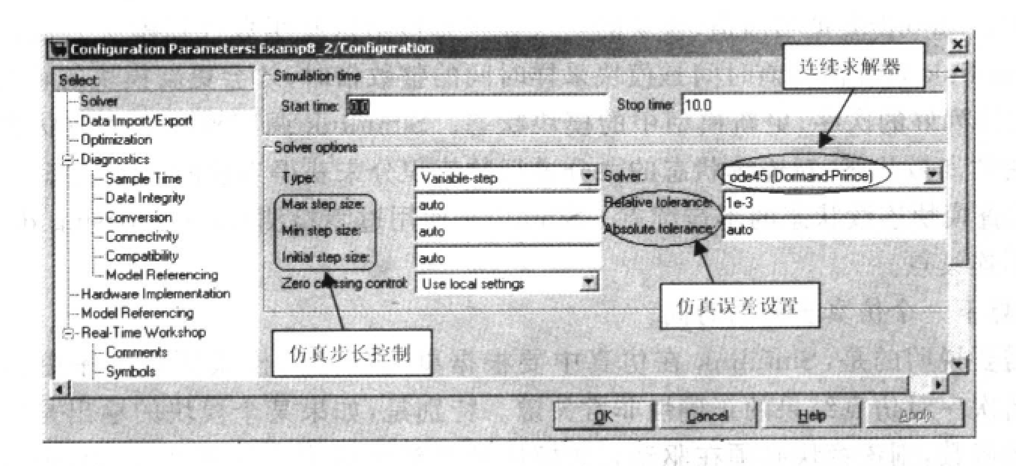


图 9.9　连续求解器设置

需要说明的一点是，实际系统很少是纯粹连续或离散的，大部分系统是混合系统。连续变步长求解器不仅考虑了连续状态的求解，也考虑了离散状态的求解，因此连续变步长求解器比较常用。连续变步长求解器首先尝试使用最大步长（仿真起始时采用初始步长）进行求解，如果在这个仿真区间内有离散状态更新，步长便减到与离散状态的更新相吻合。

9.2.3　仿真过程

Simulink 的仿真过程包括两个阶段：初始化和模型计算。

1. *初始化*

在初始化阶段，要完成的工作包括：

(1) 将模块参数传递给 MATLAB 进行估值，得到的数值结果将作为模块的实际参数。

(2) 模型的各个层次被展开。每个非条件执行子系统被它所包含的模块替代。

(3) 模型中的模块按更新的次序进行排序。排序算法产生一个列表确保具有代数环的模块在产生它的驱动输入的模块被更新后再更新。

(4) 决定模型中没有显式设置的信号属性，例如名称、数据类型、数值类型以及大小等，并且检查每个模块是否能够接收连接到它们输入端的信号。

(5) Simulink 使用属性传递的过程来决定未被设定的属性，属性传递是将源信号的属性传递到它所驱动模块的输入信号。

(6) 决定模型中所有没有显式设置采样时间的模块的采样时间。

(7) 分配和初始化用于存储每个模块的状态和输出的当前值的存储空间。

2. *模型计算*

完成初始化工作后，Simulink 就开始运行仿真了。Simulink 是使用数值积分来仿真计算的。所以，Simulink 求解器在仿真计算中起到非常重要的作用。

仿真开始时，模型首先设置待仿真系统的初始状态和输出。在每个时间步长中，Simulink 计算系统的输入、状态和输出，并更新模型来反映计算出的值。在仿真结束时，模型得出系统

的输入、状态和输出。

在每个时间步长中，Simulink 所采取的动作依次是：

(1) 按排列好的次序，更新模型中模块的输出。Simulink 通过调用模块的输出函数计算模块的输出。Simulink 把当前值、模块的输入和状态传给这些函数计算模块的输出。对于离散系统，Simulink 只有在当前时间是模块采样时间的整数倍时，才会更新模块的输出。

(2) 按排列好的次序，更新模型中的模块状态。Simulink 调用模块的离散状态更新函数来计算模块的离散状态；对连续状态的微分进行数值积分来获得当前的连续状态。

(3) 检查模块连续状态的不连续点。Simulink 使用过零检测(Zero crossing detection)检测状态的不连续点。

(4)计算下一个仿真步长的时间。

这里需要说明的是，Simulink 在仿真中要根据事先确定的模块更新次序更新状态和输出。而更新次序对仿真结果的正确性非常关键。特别是，如果某个模块的输出是它当前时刻的输入值的函数，则该模块必须在驱动它的模块被更新之后才能被更新，否则，模块的输出将无意义。

为了建立有效的更新次序，Simulink 根据输出和输入的关系，将模块分为两类。当前输出依赖于当前时刻输入的模块称为直接馈入模块，所有其他模块称为非直接馈入模块。比如，Simulink 中的 Gain，Product 和 Sum 模块是直接馈入模块，而 Constant 模块(没有输入)、Memory 模块(输出只依赖于前一个时间步长的输入)则是非直接馈入模块。基于上述的分类，Simulink 使用两个基本规则对模块进行排序：

(1) 每个模块必须在它所要驱动的所有模块中的任何一个模块更新之前被更新。这条规则确保模块在被更新时，它的输入有效。

(2) 非直接馈入模块可以按任何的次序更新，但必须在它们所要更新的直接馈入模块之前更新。这条规则可以通过把所有非直接馈入模块以任何次序放在更新列表来满足。它允许 Simulink 在排序过程中忽略非直接馈入模块。

在排序过程中，Simulink 检查和标记代数环的出现。有关代数环的概念本章将做较详细的介绍。

9.3　系统过零的概念与解决方案

Simulink 对系统仿真的控制是通过系统模型(框图)与 MATLAB 求解器的直接交互对话进行的，如图 9.7 所示。Simulink 将系统模型、模块参数传递给 MATLAB 求解器，而 MATLAB 求解器计算系统模块的输出、确定下一步仿真时间，并通过 Simulink 环境再传递给系统模型。

对话方式的核心是事件通知。系统模型通过 Simulink 仿真环境通知求解器前一个仿真步长内系统所发生的事件，以便求解器计算当前仿真时刻的结果。Simulink 用过零检测来检测系统中是否有事件发生。系统模型正是通过过零检测与事件通知完成与 MATLAB 求解器的交互的。

9.3.1 过零事件及过零检测

在系统仿真的过程中，过零是指系统模型中的信号或系统模块特征产生显著变化。这种改变包括两种情况：① 信号在上一个仿真步长中改变了符号；② 系统模块在上一个仿真时间步长中改变了模式（如积分器进入了饱和区）。

过零本身便是一个非常重要的事件，同时它也用来表示其他事件的发生，统称过零事件。Simulink 用过零来表征动态系统中的不连续性，例如系统响应的跳变等。过零事件的一个典型的示例是和地板相撞反弹的小球。要对这样的系统进行仿真，不使用过零检测，求解器不可能精确地使仿真时刻与小球和地面接触的时刻重合。这样，小球就像穿过了接触点，穿透了地板。

过零检测在检测过零事件是否发生方面发挥着重要的作用。Simulink 使用过零检测使某仿真时刻精确地（在机器精度范围内）发生在状态事件发生的时刻。因此，对于和地板相撞反弹的小球系统的仿真来说，仿真时刻可以精确地取在小球与地面接触的时刻，仿真就不会发生穿透现象，且小球的速度由负到正的转换非常迅速。Simulink 中有一个弹球的演示示例，用户可在 MATLAB 命令窗口键入 bounce 或在 MATLAB 的 demo 窗口直接找寻并打开它。感兴趣的读者也可以通过此例熟悉高级积分器的设置和使用。

9.3.2 事件通知

在系统仿真过程中，采用变步长求解器可以使 Simulink 正确地检测到系统模块与信号中过零事件的发生。如果一个模块通过 Simulink 仿真环境通知求解器在系统前一个仿真步长时间内发生了过零事件，变步长求解器就会缩小仿真步长，即使求解误差满足绝对误差和相对误差的上限要求。缩小仿真步长的目的是判断事件发生的准确时间（也就是过零事件发生的准确时刻）。虽然这样做会使系统的仿真速度变慢，但这样做对系统的某些模块是非常重要的，因为这些模块的输出表示的一个物理值的零值可能标志着系统运行状态的改变，或可能控制着另外的模块。事实上，只有少数的模块可以发出事件通知。每个模块发出专属于自己的事件通知，而且可能与不止一个类型的事件发生关联。

事件通知是 Simulink 进行动态系统仿真的核心。可以说，Simulink 动态系统仿真是基于事件驱动的。在系统仿真中，系统模型与求解器均可看做某种对象，事件通知可以理解为对象间的消息传递；对象通过消息的传递来完成系统模型和求解器之间的交互作用。

9.3.3 支持过零的模块

事实上在 Simulink 的模型库中，只有少数的模块能够产生过零事件。能够产生过零事件的模块有：Math 模型库中的求绝对值模块 Abs；最值模块 MinMax；符号运算模块 Sign；Discontinuities 模型库中的偏移模块 Backlash；死区模块 Dead Zone；交叉模块 Hit Crossing；继电模块 Relay；饱和模块 Saturation；Continuous 模型库中的积分模块 Integrator；Logic and Bit Operations 模型库中的关系运算模块 Relational Operator；Sources 模型库中的阶跃模块 Step；Subsystems 模型库中的子系统模块 Subsystem；Signal Routing 模型库中的开关模块 Switch 等。一般来说，不同模块所产生的过零事件的类型不同。例如，对于求绝对值模块 Abs，当输入改变符号时产生一个过零事件，而饱和模块 Saturation 则能够生成两个不同的过

零事件，一个用于下饱和，一个用于上饱和。

对于其他不具备过零检测能力的模块，如果需要对它们进行过零检测，则可以使用 Discontinuities 模型库中的交叉模块 Hit Crossing 来实现。当 Hit Crossing 模块的输入穿过某个偏移值(offset)时会产生一个过零事件，所以它可以用来为不具备过零能力的模块提供过零检测的能力。

一般来讲，系统模型中模块过零的作用有两种：一是用来通知求解器系统的运行模式是否发生了改变，即系统的动态特性是否发生改变；二是来驱动系统模型中的其他模块。过零信号包含 3 种类型：上升沿、下降沿、双边沿。其中，上升沿是指系统中的信号上升到零或穿过零，或者信号由零变为正；下降沿是指系统中信号下降到零或穿过零，或者信号由零变为负；双边沿是指任何信号的上升或下降沿的发生。

9.3.4　过零的举例

1. 过零点的产生与影响

例 9.1　举例说明过零的产生与影响。图 9.10 所示是系统仿真模型和仿真结果。由仿真模型可以看出系统中采用了 User-Defined Functions 模型库中的 Fcn 模块和 Math Operations 模型库中 Abs 模块。其中，Fcn 模块和 Abs 模块的输入信号分别是正弦信号和偏差为 0.5 的正弦信号。这两个模块均完成对输入信号求绝对值的功能。

由本例可以看出不支持过零事件的 Fcn 模块在求绝对值时，一些拐点被漏掉了，而支持过零事件的 Abs 模块能够使过零点处的仿真步长足够小，精确地捕获其输入信号改变符号的时刻，得到零点结果。

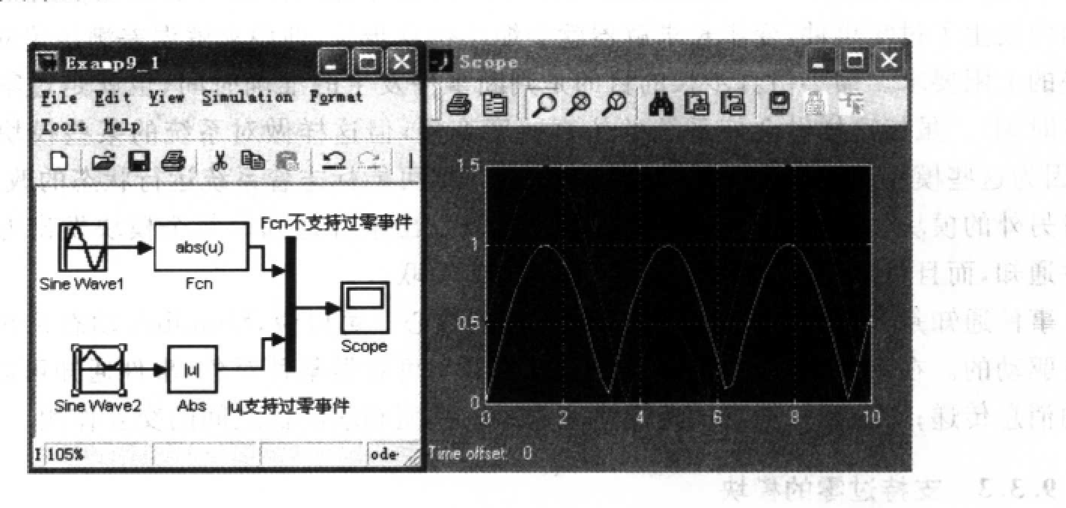


图 9.10　过零事件及其对计算结果的影响

2. 关闭过零与影响

例 9.1 中过零表示系统穿过了零点。其实，过零不仅表示信号穿过了零点，还可表示信号的陡沿和饱和。下面介绍的例 9.2 可以说明这个问题。

例 9.2　图 9.11 所示是系统仿真模型及其仿真结果。

在本例中，系统实现了输入信号由其绝对值跳变到饱和值的功能，并且其跳变过程受到仿真时间的控制。此系统中所采用的模块 Abs 和 Saturation 都支持过零事件，因此在系统的响应输出中得到了理想的陡沿。

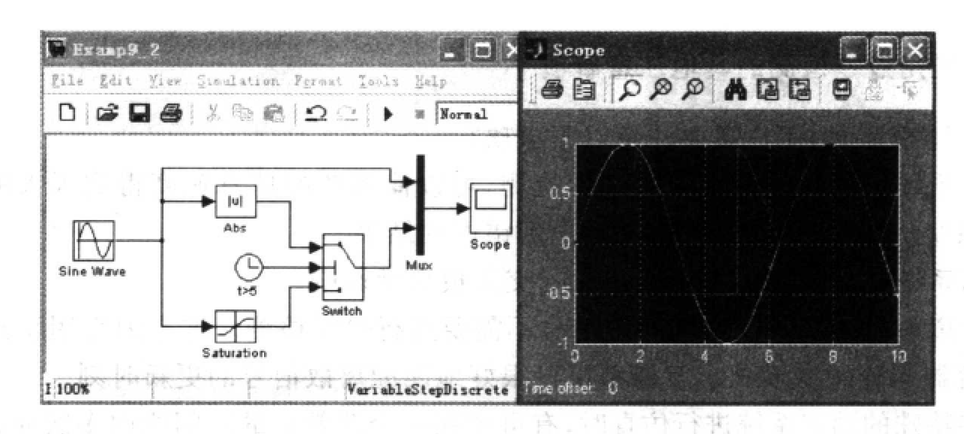
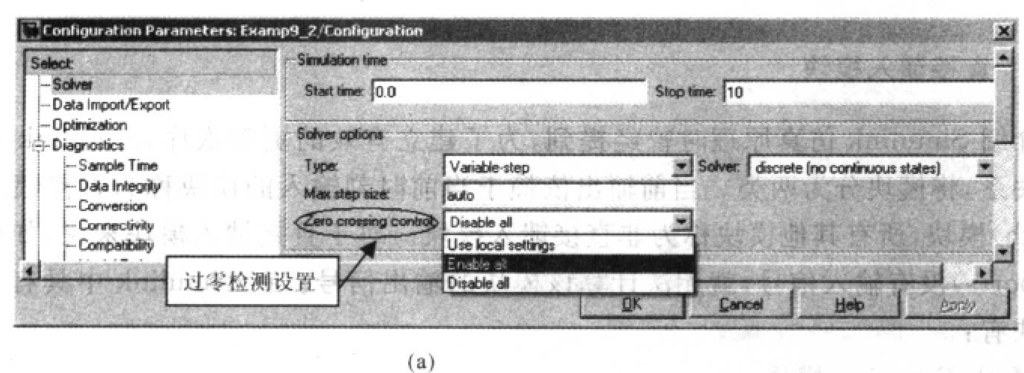
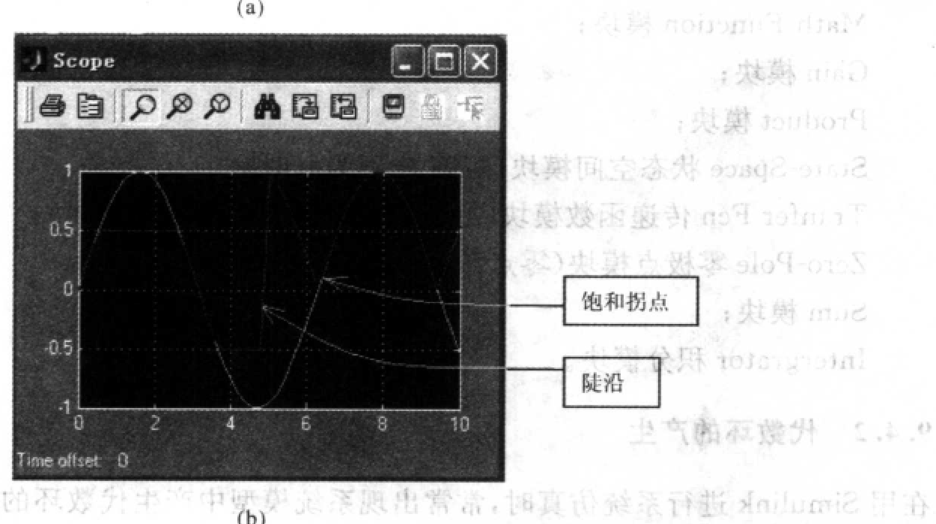


图 9.11　例 9.2 的系统仿真模型及其仿真结果

在使用 Simulink 进行动态仿真时，仿真参数默认选择使用过零检测功能。如果过零检测并不能给系统仿真带来很大的好处，用户可以关闭仿真过程中过零事件的检测功能。用户需要在 Simulation-Configuration Parameters 参数设置对话框的 Solver 选项卡中选择过零检测的开和关。图 9.12(a)，(b)所示分别是过零检测设置及关闭过零检测后，系统的仿真结果。显然，关闭过零检测功能后，系统的仿真结果在信号进入饱和时带有一些拐角，且在 5s 时的陡沿不理想。



(a)



(b)

图 9.12　例 9.2 过零检测设置及关闭过零检测后系统仿真结果

(a) 系统过零检测设置；　(b) 关闭过零检测后系统仿真结果

9.3.5 使用过零检测的其他注意事项

在使用过零检测时，用户需要注意以下事项：

(1) 关闭系统仿真参数设置中的过零检测，可以使系统的仿真速度得到很大的提高。但可能会引起系统仿真结果的不精确，甚至出现错误的结果。

(2) 关闭系统过零检测对 Hit Crossing 交叉模块无影响；

(3) 对于离散模块及其产生的离散信号不需要进行过零检测。这是因为用于离散系统仿真的离散求解器与连续变步长求解器都可以很好地匹配离散信号的更新时刻。

在对某些特殊的动态系统进行仿真时，有可能在一个非常小的时间段内多次通过零点。这将导致在同一时间内多次探测到信号的过零，从而使得 Simulink 仿真终止。在这种情况下，用户应该关闭过零检测功能再进行仿真。但是，对于模块过零非常重要的系统，用户可以采用在系统模型中串接交叉 Hit Crossing 模块，并关闭过零检测功能的方法来实现过零的检测。

9.4 系统代数环的概念与解决方案

9.4.1 直接馈入模块

前面介绍 Simulink 仿真原理时曾经提到，为了建立有效的更新次序，Simulink 根据输出和输入的关系，将模块分为两类。当前输出依赖于当前时刻输入的模块称为直接馈入(Direct feedthrough)模块，所有其他模块称为非直接馈入模块。对于直接馈入模块来说，如果输入端口(Input ports)没有输入信号，就无法计算该模块的输出信号。在 Simulink 中具有直接馈入特性的模块有：

Math Function 模块；
Gain 模块；
Product 模块；
State-Space 状态空间模块(矩阵 ***D*** 不为 0 时)；
Tranfer Fcn 传递函数模块(分子和分母多项式阶次相同时)；
Zero-Pole 零极点模块(零点和极点数目相同时)；
Sum 模块；
Intergrator 积分模块。

9.4.2 代数环的产生

在用 Simulink 进行系统仿真时，常常出现系统模型中产生代数环的提示。在下列两种情况下，系统模型中会产生代数环：

第一，具有直接馈入特性的模块的输入端口直接由此模块的输出驱动；

第二，具有直接馈入特性的模块的输入端口由其他具有直接馈入特性的模块所构成的反馈回路间接地驱动。

图 9.13 所示是非常简单、非常典型的代数环的产生示例。

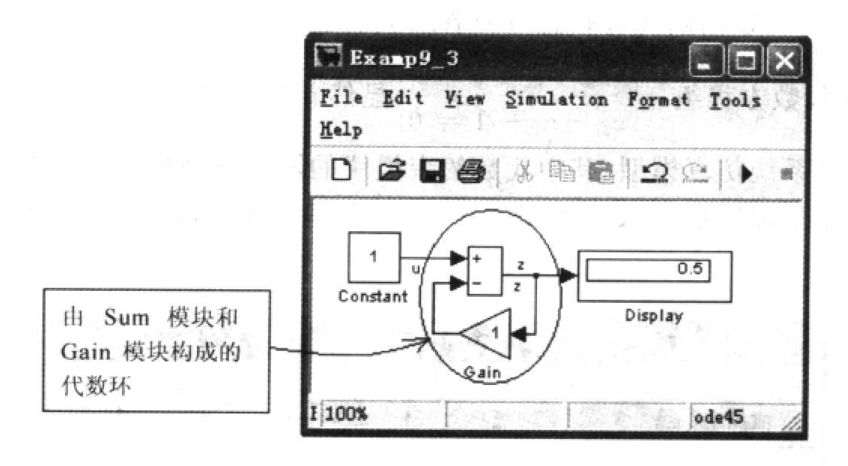


图 9.13　具有代数环的系统模型

运行该仿真模型时，MATLAB 命令窗口会提示：

Found algebraic loop containing block(s)：

'Examp9_3/Gain'

'Examp9_3/Sum' (algebraic variable)

即系统中存在代数环，且代数环包含 Sum 和 Gain 模块。从仿真图中可以看出此代数环回路由一个求和模块和一个增益模块构成。其中模块 Sum 的输出状态 z 同时又作为该模块的输入。由于求和模块具有直接馈入的特性，即模块的输出直接依赖于模块的输入，因而构成了代数环。很显然，此代数环可以用数学表达式 $z=u-z$ 直接描述，相应的其输出状态为 $z=u/2$。但对于大多数的代数环系统而言，难以通过直接观察来求解。

如果系统中出现了代数环，由于代数环的输入和输出之间是相互依赖的，组成代数环的所有模块都要求在同一个时刻计算输出。这与系统仿真的顺序概念不符，因此，最好使用其他的方法来解决代数环的求解问题。

9.4.3　代数环的举例与解决方案

对于系统中所产生的代数环，解决的方法有 3 种：

第一，使用手工的方法对系统方程直接求解；

第二，对代数环进行代数约束；

第三，切断代数环。

1. *代数环解决方法一：使用手工的方法对系统方程直接求解*

例 9.3　代数环的直接求解。

图 9.13 是例 9.3 的系统仿真模型。很显然，此代数环可以用数学表达式 $z=u-z$ 直接描述，其输出状态为 $z=u/2$，当输入 $u=1$ 时，计算输出 $z=0.5$。其结果见 Display 模块所示。事实上，Simulink 中有一个内置的代数环求解器，可以对含有代数环的简单的系统模型进行正确的计算(见图 9.13)。

2. *代数环解决方法二：使用代数约束*

系统模型中使用代数约束 Algebraic Constraint 模块并给出约束初值，可以很方便地对代

数方程进行求解。代数约束模块的输入 $F(z)$是一个代数表达式，输出是模块的代数状态。代数约束模块通过调整其输出的代数状态以使其输入为零。

例 9.4 求解代数方程组：$\begin{cases} z_1 + z_2 - 1 = 0 \\ z_2 - z_1 - 1 = 0 \end{cases}$，显然此方程组的解为 $z_1 = 0$，$z_2 = 1$。

图 9.14 所示是系统仿真模型，其中代数约束模块的输出分别是代数状态 z_1 和 z_2。z_1 和 z_2 分别通过反馈回路作为代数约束模块的输入。其仿真结果如图 9.14 中 Display 模块中所显示。

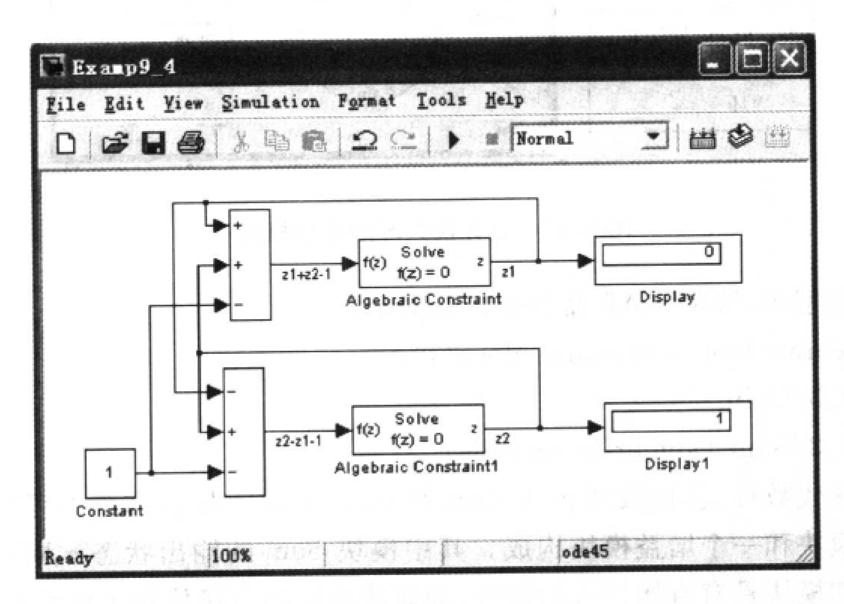


图 9.14 使用代数约束解决代数环问题

当系统中使用代数约束时，系统中将出现代数环。对于系统中存在代数环的系统，Simulink 会在每个仿真步长中调用代数环求解器对系统进行求解。代数环求解器通过迭代的方法对系统进行求解，由于对系统的求解使用了迭代方法，因而，含有代数环的系统的仿真速度相对不含代数环的系统要慢一些。

在使用代数约束模块时，Simulink 使用牛顿迭代法求解代数环。虽然采用这种方法是一种稳定的算法，但是如果代数状态的初始值选择得不合适，算法可能不收敛。因此，用户在使用代数约束时，一定要注意代数约束模块输出的代数状态的初始值的选取问题，如果初始值选取的不同，有可能造成最终结果的不同。下面举例说明这个问题。

例 9.5 使用代数约束求解方程

$$x^2 - x - 2 = 0 \quad \text{即} \quad (x+1)(x-2) = 0$$

的根（显然此方程的根是 $x_1 = -1, x_2 = 2$）。

图 9.15 所示是求解该方程的系统仿真模型。如果其中的代数约束模块的初始值分别取为 5 或 −5，则仿真得出的结果是不同的。不同的结果见图 9.15 中 Display 和 Display1 模块的显示。

3. 代数环解决方法三：切断环

在实际应用时，前面介绍的两种解决含代数环系统的仿真问题的方法有时是不方便的。这是因为：第一，很多情况下，很难甚至不可能进行手工求解；其次，在使用代数约束，由 Simulink内置的代数环求解器对含代数环的系统进行仿真时，虽然系统可以有效地求解代数

环，但由于采用的牛顿迭代方法需要在每个仿真步长内进行多次迭代，因此，仿真速度会大幅度地降低。因此，用户可以通过某种破坏代数环产生条件的方式来切断模型中的代数环结构，从而加快系统的仿真速度。

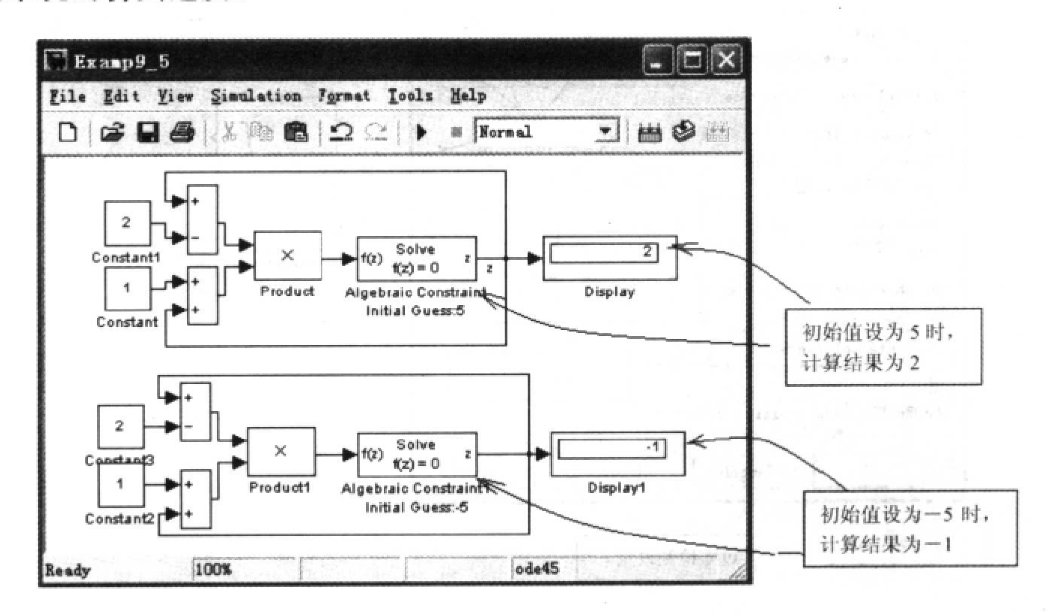


图 9.15　代数约束状态初始值设置对计算结果的影响

常用的切断代数环的方法是在代数环中加入 Discrete 模型库中的存储模块（Memery 模块）或单位延迟模块（Unit Delay 模块）。尽管使用这种方法非常容易，但是在一般条件下并不推荐这样做，因为加入存储或延迟模块会改变系统的动态特性，而且对于不适当的初始估计值，有可能导致系统不稳定。

9.5　高级积分器

积分运算是动态系统仿真中常见的运算之一。在使用 Simulink 对实际的动态系统进行仿真计算时，积分运算是构成 Simulink 求解器的核心技术之一。前面在举例介绍动态系统的仿真方法时，仅使用了简单的积分器，用户仅仅需要设置积分器的初始值，其他均采用积分器的缺省设置。本节将简单介绍高级积分器的概念、设置方法和应用。

图 9.16 是默认参数设置下的积分器外观和选择所有参数设置后积分器的外观比较。前者俗称简单积分器，后者为高级积分器。要想正确灵活地使用选择所有参数设置后的积分器，必须先了解这种积分器各个端口的含义和设置。下面首先介绍积分器参数设置对话框。积分器参数设置对话框如图 9.17 所示。

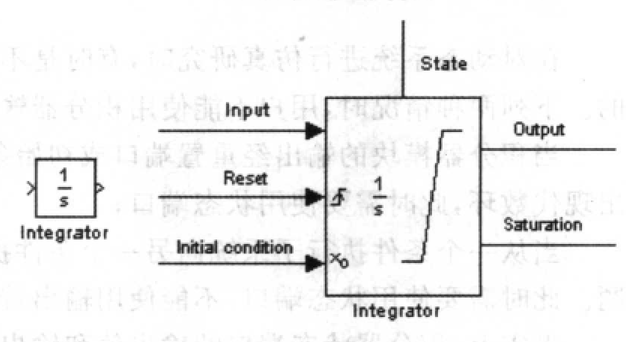


图 9.16　积分器外观比较

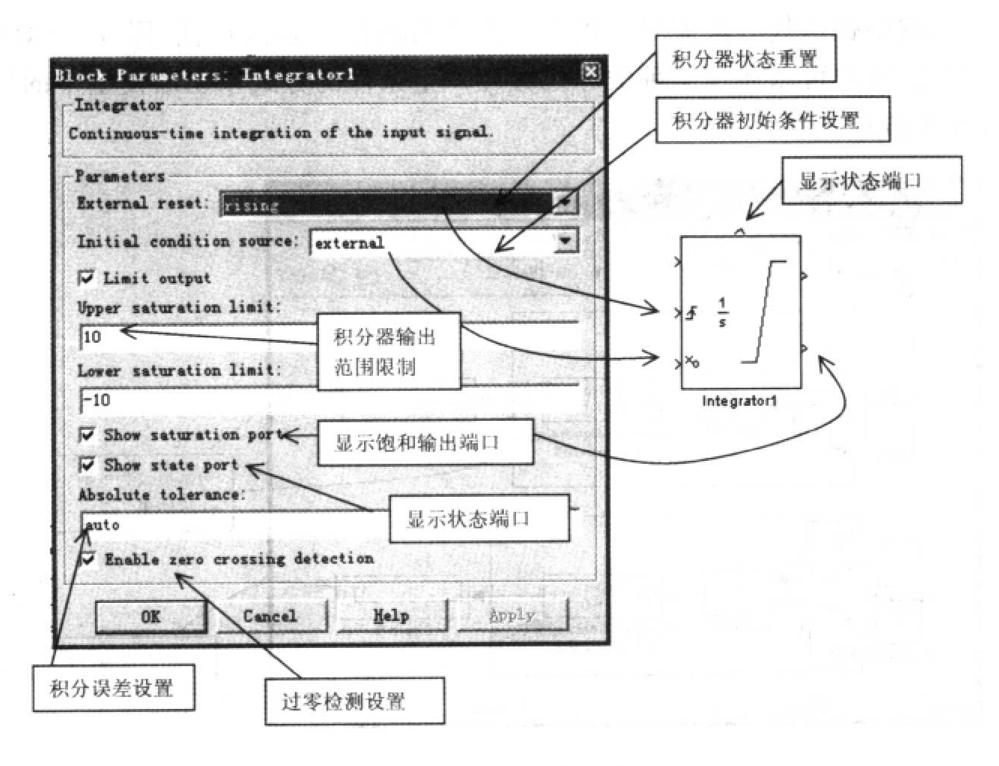


图 9.17 高级积分器设置

9.5.1 积分器初始条件端口

设置积分器初始条件的方法有两种：

(1) 内部输入源设置：点击积分器模块，打开积分器参数设置对话框，在初始条件源设置(Initial condition source)中选择内部设置(Internal)，并在下面的文本框中键入给定的初始条件即可。这种设置方式不显示积分器初始条件设置端口 x_0。

(2) 外部输入源设置：在初始条件源设置中选择外部设置(External)，初始条件设置端口以 x_0 为标志显示。这种设置需要使用 Signal Attributes 模型库中的 IC 模块设置积分器初始值。

9.5.2 积分器状态端口

在对动态系统进行仿真研究时，有时是不能使用积分器输出端口，而需要使用其状态端口的。下列两种情况时，用户不能使用积分器输出端口，而必须使用状态端口：

当积分器模块的输出经重置端口或初始条件端口反馈至模块本身时，会造成系统模型中出现代数环，此时需要使用状态端口；

当从一个条件执行子系统向另一个条件执行子系统传递状态时，可能会引起时间同步问题。此时需要使用状态端口，不能使用输出端口。

事实上，积分器状态端口的输出值和输出端口的输出值本身没有大的区别，其不同之处仅在于二者产生的时间略微有所不同。这正是 Simulink 解决上述问题的方案。选择 Show state port 复选框，状态端口会显示在积分器的顶部，如图 9.17 所示。

9.5.3　积分器输出范围限制和饱和输出端口(Saturation)

在对动态系统进行仿真的过程中，所使用的积分器的输出可能会超过系统本身所允许的上限或下限值，选择积分器输出范围限制框(Limit output)，并设置上限值(Upper saturation limit)与下限值(Lower saturation limit)，可以将积分器的输出限制在一个给定的范围之内。此时积分器服从下列规则：

(1) 当积分结果小于或等于下限值且输入信号为负，积分器输出保持在下限值(下饱和区)；

(2) 当积分结果介于下限值和上限值之间时，积分器输出为实际积分值；

(3) 当积分结果大于或等于上限值且输入信号为正，积分器输出保持在上限值(上饱和区)；

选择 Show saturation port 复选框可以在积分器上显示饱和端口 Saturation。饱和端口的输出用来表示积分器的饱和状态，其取值有以下 3 种情况：

(1) 输出为 1，表示积分器处于上饱和区；

(2) 输出为 0，表示积分器处于正常范围之内；

(3) 输出为 −1，表示积分器处于下饱和区。

当选择输出信号限制时，积分器模块将产生 3 个过零事件：一个用来检测积分结果何时进入上饱和区，一个用来检测积分结果何时进入下饱和区，还有一个用来检测积分器何时离开饱和区。

9.5.4　积分器重置

选择积分器状态重置框可以重新设置积分器的状态，其值由外部输入信号决定。此时在积分器输入端口下方出现重置触发端口。可以采用不同的触发方式对积分器状态进行重置：

(1) 上升沿触发重置方式：选择 rising；

(2) 下降沿触发重置方式：选择 falling；

(3) 双边沿触发重置方式：选择 either。

当重置信号非零时，选择 level 重置积分器状态，并使积分器输出保持在初始状态。

积分器的重置端口具有直接馈入的特性。积分器的输出不管是直接反馈还是通过其他具有直接馈入特性的模块反馈至其重置端口，都会使系统出现代数环，而使用状态端口代替输出端口则可以避免代数环的产生。

9.5.5　积分器绝对误差设置(Absolute tolerance)

默认情况下，积分器采用 Simulink 自动设置的绝对误差限。用户也可以根据自己的需要设置积分器的绝对误差限，直接在 Absolute tolerance 下键入误差上限即可。

9.5.6　过零检测设置

当选择过零检测功能时，积分器会自动产生过零事件。

下面使用例 9.6 说明高级积分器的应用。

例 9.6　小球撞击地板系统的仿真模型及其积分器 Velocity 参数设置窗口如图 9.18

所示。

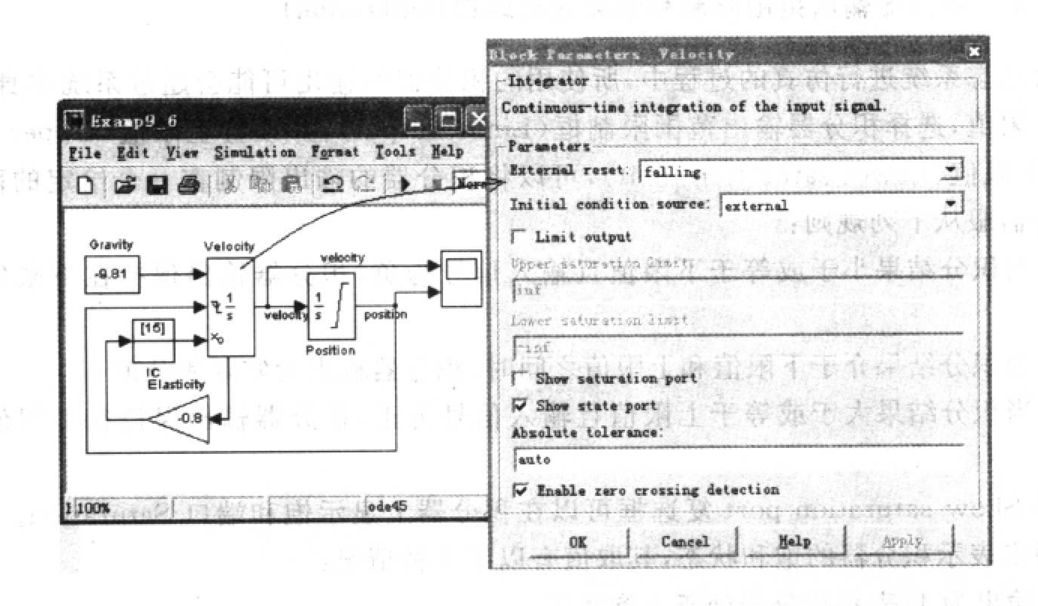


图 9.18 bounce 仿真模型及其 Velocity 积分器设置

小球撞击地板系统的仿真模型模拟以一定初速度向上弹的小球反复与地面相撞，其位置逐渐发生变化并最终变为零的过程。仿真模型不考虑空气的阻力。这样，小球在空中运行过程中的机械能量保持不变，保证了小球在与地面撞击后的瞬时速率与下次小球与地面撞击前的速率相同；其次，仿真模型对小球和地面的撞击过程做了简化。模型假设撞击后的速度与撞击前的速度之比始终是个常数，本例中该常数取值为－0.8。

小球在空中做自由落体运动。根据牛顿运动学定律，小球的运动方程为

$$g=\frac{\mathrm{d}v}{\mathrm{d}t} \text{ 和 } v=\frac{\mathrm{d}x}{\mathrm{d}t}$$

其中，g 为重力加速度，v 为小球速度，x 为小球在空间的位置。所以在构建仿真模型时，需要两个积分器：Position 和 Velocity。对于积分器 Position 模块，它的输入是速度信号，且小球的初始位置是 10，所以其初始状态设置为 10。此外，球与地板接触与否对位移的运算规律没有任何影响，所以 Position 积分器的积分限是 0 至无穷大。但对于 velocity 积分模块，由于小球与地板接触之后使小球速度方向和积分初始值均发生了变化，因此必须对此积分器进行参数设置。velocity 积分模块的参数设置情况如图 9.18 所示。

Velocity 积分模块选择了积分器状态重置功能，采用位置信号的下降触发标志使积分重新回到初始值。而 Initial condition source 参数设置为 external，则积分的初始值由外部输入，其初始值由一个初值为 15 的 IC 模块给定。Show state port 复选框选中的结果是用户可以使用积分器的输出积分状态通过具有直接馈入特性的 Gain 模块输入到 IC 模块，而不用直接使用积分器的输出以避免系统中出现代数环。该积分器参数设置时选择了过零检测功能，使得 Simulink 能够通过过零检测来准确地确定小球与地板接触的时刻。一旦小球位移从正变为负就产生一个下降触发事件，Simulink 通过过零检测来捕获这个事件。一旦检测到下降沿，就会使 Velocity 重新设置为初始值，而此时的初始值为当前状态值乘以－0.8。运行 Bounce 模型的仿真结果如图 9.19 所示。

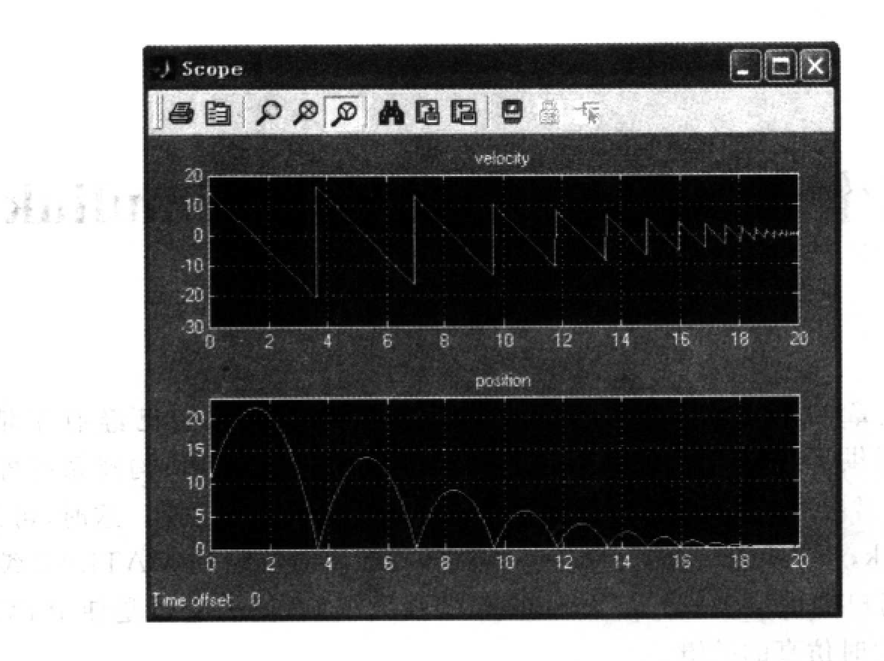


图 9.19　Bounce 模型的仿真结果

习　题

9.1　使用代数约束求解方程 $x^2+3x-4=0$。

9.2　求解代数方程组 $\begin{cases} z_1+z_2=5 \\ z_2-z_1=1 \end{cases}$，并将结果显示在 Display 模块中。

9.3　建立图 9.20 所示的仿真模型，熟悉高级积分器的使用方法，分析其所能完成的功能。

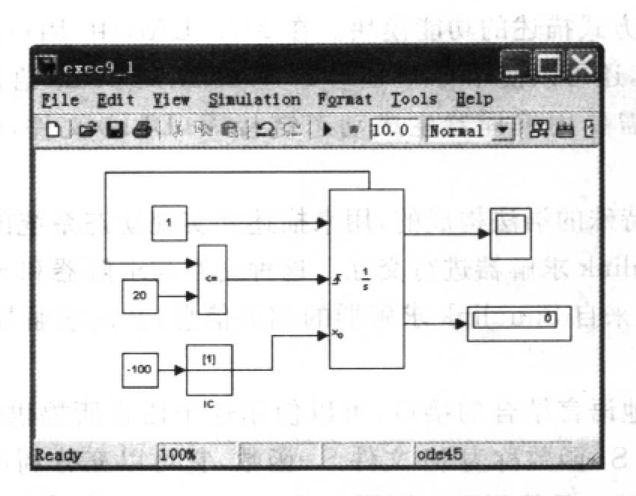


图 9.20　题 9.3 图

第十章　用S-函数扩展Simulink

通过第八、九章的学习，用户对用 Simulink 建模的基本思想已经有了清晰的认识，Simulink为用户提供了许多的内置库模块，用户只须使用这些库模块构建系统即可。但在实际应用中，用户通常会发现有些过程用 Simulink 的库模块不容易建模。这时，可以使用 S-函数来扩展 Simulink。S-函数结合了 Simulink 框图图形化的特点和 MATLAB 编程灵活方便的优点，从而给用户提供增强和扩展 Simulink 的强大机制，同时它也是使 RTW(Real-Time Workshop)实现实时仿真的关键。

10.1　S-函数概述

10.1.1　S-函数的基本概念

S-函数是 System function 系统函数的简称，是指采用非图形化(即计算机语言，而非 Simulink 系统模块)的方式描述的功能模块。在 MATLAB 中，用户除了可以使用 MATLAB 代码编写 S-函数以外，还可以使用 C，C++，FORTRAN 或 Ada 语言编写 S-函数，只不过用这些语言编写程序时需要用编译器生成动态链接库(DLL)文件，然后在 Simulink 中直接调用。

S-函数是由一种特殊的语法构成的，用来描述并实现动态系统的。它采用一种特殊的调用语法，使函数和 Simulink 求解器进行交互。这种交互与求解器和 Simulink 仿真模型间的交互相类似：S-函数接受来自 Simulink 求解器的相关信息，并对求解器发出的命令做出适当的响应。

S-函数作为与其他语言结合的接口，可以使用这个语言所提供的强大功能。例如，使用 MATLAB 语言编写的 S-函数称为 M 文件 S-函数，它可以充分利用 MATLAB 所提供的丰富资源，方便地调用各种工具箱函数和图形函数；而使用 C 语言编写的 S-函数被称为 C MEX 文件 S-函数，则可以实现对操作系统和外部设备等的访问，也可以提供与操作系统的接口。另外，S-函数可以使用其他多种语言编写，因此可以实现代码的移植，即将已有的代码结合进来，而不须在 Simulink 中重新实现算法。

S-函数中采用非图形化的方式描述系统，其内部采用文本方式输入描述系统的公式、方程，这种方式非常适合复杂动态系统的数学描述，且可以在仿真过程中对仿真进行精确的控制。

10.1.2　如何使用S－函数

在动态系统仿真中，要想将S－函数加入Simulink仿真模型中，用户需要将User Defined Functions模型库中的S－Function模块拖进该模型中。S－Function模块是一个单输入单输出模块，如果有多个输入与输出信号，用户需要使用Mux模块和Demux模块对信号进行组合或分离。S－Function模块仅仅是以图形的方式提供给用户一个S－函数的使用接口，在它的参数设置对话框中仅包含S－函数的名称及函数所需的参数列表（见图10.1），而S－函数所实现的功能则由S－函数源文件描述，S－函数源文件必须由用户自行编写。

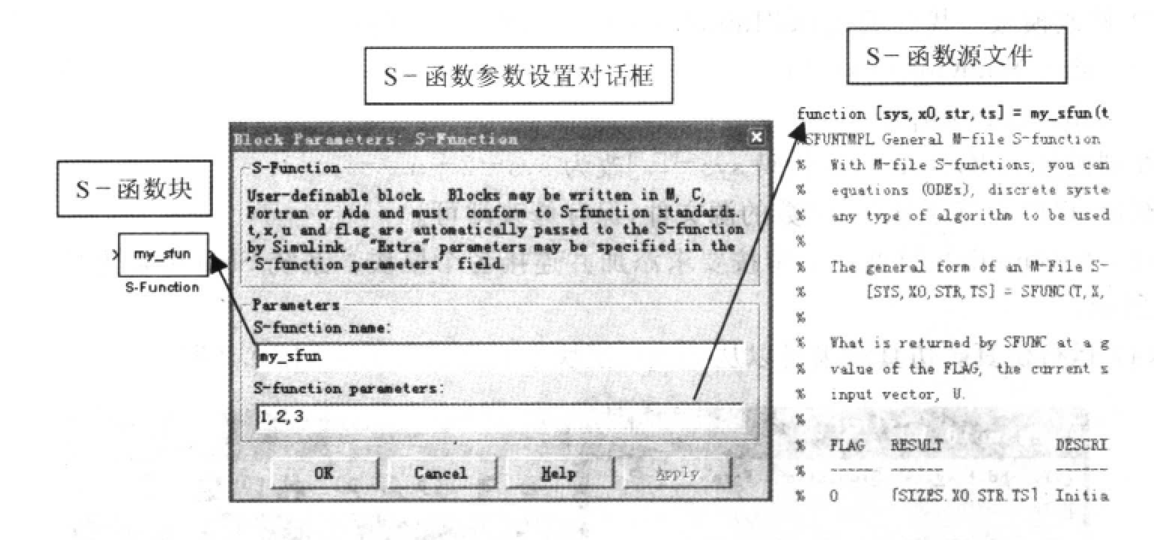


图10.1　S－函数模块、参数设置对话框及其源文件的关系

使用S－函数的步骤如下：

（1）在系统的Simulink仿真框图中添加S-function模块，并进行正确的设置。

（2）创建S－函数源文件。创建S－函数源文件的方法有多种，用户可以按照S－函数的语法格式自行编写代码，但是这样做很麻烦，且容易出错。Simulink在S－function Examples模型库中为用户提供了针对不同语言的很多S－函数模板和例子，用户可以根据自己的需要修改相应的模板或例子即可完成S－函数源文件的编写工作。

（3）在系统的Simulink仿真框图中按照定义好的功能连接输入输出端口。

这里需要说明的是，S－function模块中S－函数名称必须和用户建立的S－函数源文件的名称完全相同，S－function模块中的S－函数参数列表必须按照S－函数源文件中的参数顺序赋值，且参数之间需要用逗号隔开。另外，用户也可以使用子系统封装技术对S－函数进行封装，这样做的好处是可以增强系统模型的可读性。

为了方便用户编写C MEX S-函数，Simulink的User Defined Functions模型库中为用户编写C MEX S－函数提供了一个图形化的集成的图形开发环境S－Function Builder。用户只需在S－Function Builder中相应的位置写入相应的名称和代码即可编译成相应的MEX文件。

为了使读者尽快掌握S－函数的使用步骤，先举一个简单的例子。

例10.1　使用S－函数实现系统：$y=2*u$。

解　（1）在Simulink模型框图中添加S－Function模块，打开S－Funfction模块的参数

设置对话框，参数 S - Function name 需设置为 timestwo。

(2) 创建 S-函数源文件。

1) 打开 M 文件 S-函数模板文件 Sfuntmpl. m，并在指定目录下另存为 timestwo. m。打开 M 文件 S-函数模板文件 Sfuntmpl. m 的方法有两种：

方法一，在 MATLAB 命令窗口键入 edit S - funtmpl 即可；

方法二，在 Simulink 浏览器中寻找 S - Function demos 模型库，其中包含各种语言编写的 S-函数例子和模板。对于本例，用户只需点击 M - file S - Functions 就可以看到 M 文件 S-函数模板文件 Sfuntmpl. m 和几个编程例子，双击模板文件 Sfuntmpl. m 即可。

2) 修改模板。找到函数 mdlInitializeSizes，修改以下代码：

```
size. NumOutputs=1;
size. NumIutputs=1;
```

找到函数 mdOutputs，将代码 sys=[ ]改为 sys=2 * u

至此，已经写好了该 S-函数的源文件，保存修改即可。

(3) 在 Simulink 模型框图中按要求添加并连接各个模块。系统 Simulink 仿真模型如图 10. 2 所示。

(4) 运行仿真。仿真结果可以从 Scope 模块中观察。结果如图 10. 2 所示。

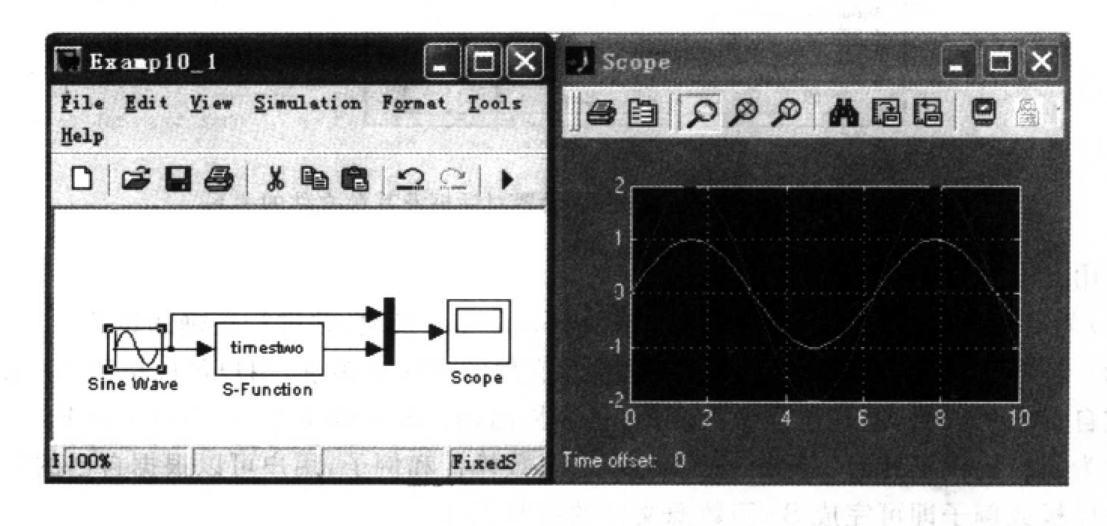


图 10. 2 例 10. 1 系统 Simulink 仿真模型及其结果

10. 1. 3 与 S-函数相关的术语

用户对 S-函数相关的术语的理解对于了解 S-函数的工作原理、编写 S-函数源文件是非常有用的。

1. *仿真例程*(Routines)

Simulink 在仿真的不同阶段调用 S-函数的不同的功能函数以完成不同的任务。对于 M 文件 S-函数，Simulink 通过传递一个 flag 参量给 S-函数，通知 S-函数当前所处的仿真阶段，以便执行相应的功能函数。S-函数的功能函数包括初始化、计算输出、更新离散状态、计算导数、结束仿真等。这些功能函数称为仿真例程或回调函数(call back function)。在写 M 文件 S-函数时，用户只需用 MATLAB 语言为每个 flag 对应的功能函数编写代码即可。表

10.1列出了各仿真阶段的功能函数及其对应的flag值。

表10.1　各仿真阶段的仿真例程及其flag值

仿真阶段	S-函数仿真例程(回调函数)	Flag值(M文件S-函数)
初始化	mdlInitializeSizes	0
计算下一个采样点	mdlGetTimeofNextVarHit	4
计算输出值	mdlOutputs	3
更新离散状态	mdlUpdate	2
计算导数	mdlDerivatives	1
结束仿真	mdlTerminate	9

2. 直接馈入(Direct Feedthrough)

直接馈入是指模块的输出或采样时间(变速率模型)直接由某个输入端口控制。判断一个S-函数是否具有直接馈入的标准是:某时刻系统的输出y包含该时刻系统的输入u,即计算系统输出的方程中包含输入变量u;若系统是一个变采样时间系统,且下一个采样点的计算与输入u有关。

馈入标志的设置不仅关系到系统模型中的系统模块的执行顺序,而且关系到对代数环的检测和处理。因此正确设置馈入标志是非常重要的。

3. 采样时间和偏移量(Sample time & offsets)

M文件和C MEX文件S-函数都允许用户十分方便地设定S-函数被调用的时间。

采样时间在离散系统中控制采样点的间隔,偏移量则用于延迟采样点。一个采样点对应的时间值由下列公式计算:

$$\text{TimeHit} = n \times \text{period} + \text{offset}$$

式中,n表示当前仿真步,是整数。

如果用户定义了一个离散采样时间,Simulink就会在所定义的每个采样点调用S-函数的mdlOutputs和mdlUpdate例程。

对于连续时间系统,采样时间和偏移量的值均应设置为零。采样时间还可以继承来自驱动模块、目标模块或系统最小的采样时间,这种情况下,采样时间值设置为−1。

4. 动态输入(Dynamically sized inputs)

S-函数支持动态可变维数的输入。S-函数输入变量的维数取决于驱动S-函数模块的输入信号维数。仿真开始时,通过size或length函数确定输入信号的维数。然后就可以利用这个维数来估计连续状态数目、离散状态数目和输出向量的维数。

在M文件S-函数中动态设置输入维数时,应该把sizes数据结构的对应成员设置为−1。

比如在例10.1中,如果将函数mdlInitializeSizes中的代码设置成size.NumOutputs=−1; size.NumIutputs=−1; 则当输入是两维信号,幅值分别为1和3的正弦信号,系统的输出也是两维信号,结果如图10.3所示。

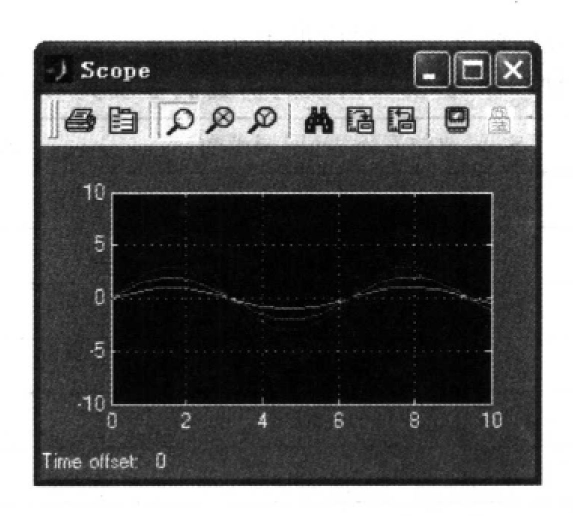


图 10.3　例 10.1 进行动态输入设置后的仿真结果

10.2　S-函数工作原理

了解 S-函数的工作原理对于用户掌握 S-函数的编写方法是非常有用的，对用户对于 Simulink 的仿真原理的理解也是很有帮助的。本节介绍 S-函数的工作原理。

在具体介绍 S-函数的工作原理之前，首先需要回顾一下 Simulink 模块的工作原理。

Simulink 中的每个模块都有 3 个基本元素：输入向量、状态向量和输出向量，分别表示为 u，x 和 y。图 10.4 反映了它们之间的关系。在 Simulink 模块的三个元素中，状态向量是最重要的，也是最灵活的概念。在 Simulink 中状态向量可以分为连续状态、离散状态或两者的结合。输入、输出及状态的关系可以用状态方程描述：

输出方程：　$y = f_o(t, x, u)$

连续状态方程：　$dx = f_d(t, x, u)$

离散状态方程：　$x_{k+1} = f_u(t, x, u)$

输入 (u) → 状态 (x) → 输出 (y)

模块

图 10.4　Simulink 模块的基本模型

Simulink 在仿真时将上述方程对应不同的仿真阶段，它们分别是计算模块的输出、更新离散状态、计算连续状态的微分。在仿真开始和结束，还包括初始化和结束仿真两个阶段。在每个阶段，Simulink 都反复地调用模块。

至此，读者已经接触到了几个关于仿真的概念：仿真步长（Simulation step）、仿真阶段（Simulation stage）。为了深入了解 S-函数的工作原理，还需了解一个概念：仿真循环（Simulation loop）。一个仿真循环就是由仿真阶段按一定顺序组成的执行序列。对于每个模块，经过一次仿真循环就是一个仿真步长，而在同一个仿真步长中，模型中各模块的仿真按照事先排好的顺序依次执行。这个过程可以用图 10.5 表示。

从图中可以看出，在仿真开始时，Simulink 首先对模型进行初始化，此阶段不属于仿真循环。在所有模块都初始化后，模块进入仿真循环，在仿真循环的每个阶段，Simulink 都要调用模块或者 S-函数。由于在积分时，对仿真步长有要求，所以此时需要将仿真步长细化。完成一个仿真循环就进入下一个仿真步长，如此循环直至仿真结束。

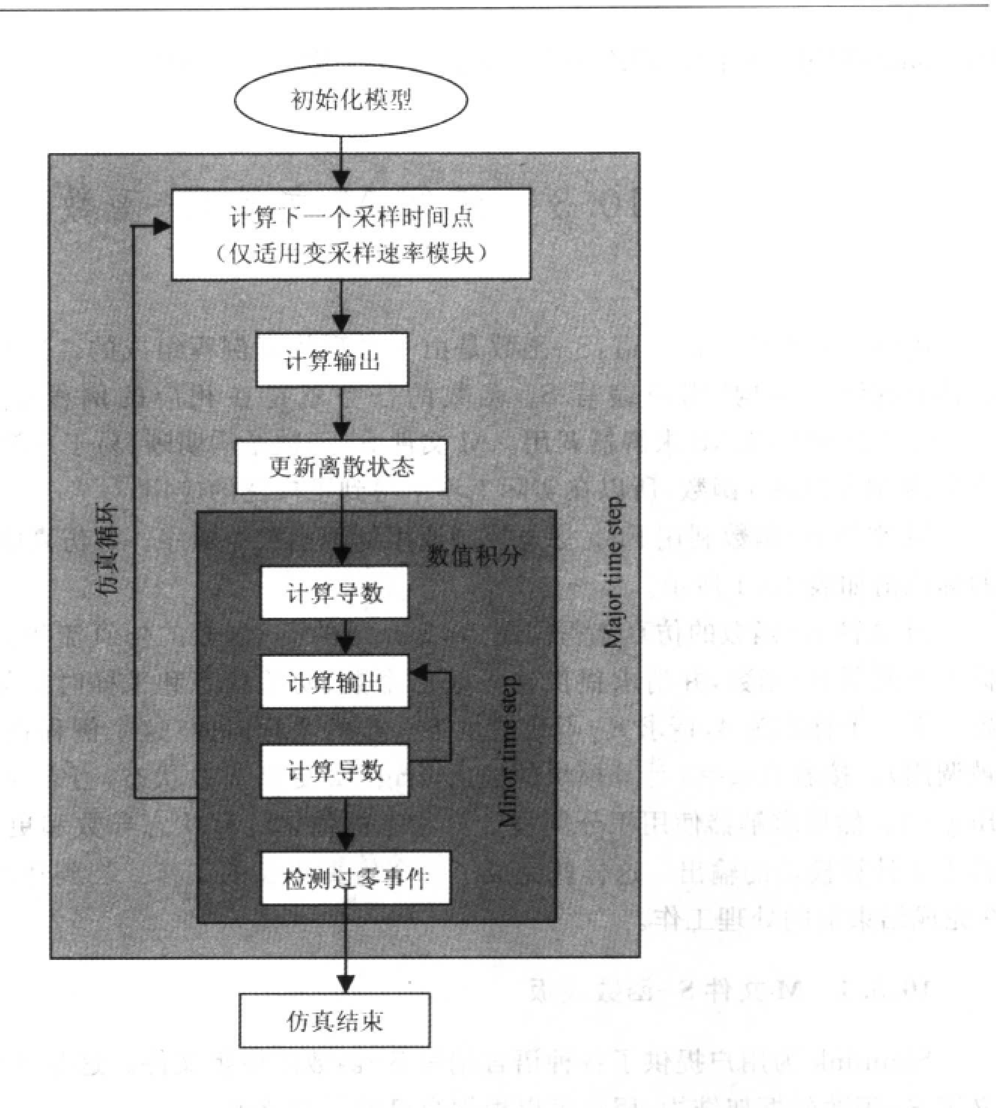


图 10.5　S-函数仿真流程

在调用模型中的S-函数时，Simulink会调用用户定义的S-函数的例程来实现每个仿真阶段要完成的任务。这些任务包括：

（1）初始化：仿真开始前，Simulink在这个阶段初始化S-函数，完成的主要工作包括：

1）初始化包含S-函数所有信息的结构体SimStruct；

2）确定输入输出端口的数目和大小；

3）确定模块的采样时间；

4）分配内存和Sizes数组。

（2）计算下一个采样时刻。如果模型使用变步长求解器，那么就需要在当前仿真步长内确定下一个采样点的时间，也即下一个仿真步长的大小。

（3）计算输出：计算所有输出端口的输出值。

（4）更新离散状态：此例程在每个仿真步长处都要执行一次，为当前时间的仿真循环更新离散状态；

（5）数值积分：这个阶段只有模块具有连续状态和非采样过零点时才会存在。如果S-函数存在连续状态，Simulink就在细化的小时间步长中调用S-函数的输出（mdlOutputs）和微分（mdlDerivatives）例程。如果存在非采样过零点，Simulink将调用S-函数中的输出（md-

lOutputs)和过零检测(mdlZeroCrossngs)例程,以定位过零点。

10.3 编写 M 文件 S-函数

由前面的介绍可以知道,S-函数是由一系列仿真例程组成的。这些仿真例程就是 S-函数特有的语法结构,用户编写 S-函数的任务就是在相应的例程中填写适当的代码,供 Simulink及 MATLAB 求解器调用。M 文件 S-函数结构明晰,易于理解、书写方便,可以调用丰富的 MATLAB 函数,所以在实际工作中得到了广泛的应用。

M 文件 S-函数利用 Flag 标志控制调用例程函数的顺序。各仿真阶段的仿真例程及对应的标志值如表 10.1 所示。

M 文件 S-函数的仿真流程同图 10.5 介绍的 S-函数的仿真流程。在初始化阶段,通过标志 0 调用 S-函数,并请求提供输入输出个数、初始状态和采样时间等信息。然后,仿真开始。下一个标志为 4,请求 S-函数提供下一步的采样时间(这个例程在单采样速率系统下不被调用)。接着 flag=3 计算模块的输出,flag=2 更新离散状态,当需要计算连续状态导数时 flag=1。然后求解器使用积分例程计算状态的值。计算状态导数和更新离散状态之后通过标志 3 计算模块的输出。这样就完成了一个仿真步长的工作。当到达结束时间时,采用标志 9 完成结束前的处理工作。

10.3.1 M 文件 S-函数模板

Simulink 为用户提供了各种语言编写 S-函数的模板文件。这些 S-函数的模板文件中定义了 S-函数的框架结构,用户可以根据自己的需要修改。

编写 M 文件 S-函数时,需要使用 M 文件 S-函数模板文件 sfuntmpl.m 文件。该文件包含了所有的 S-函数的例程,即包含 1 个主函数和 6 个子函数。在主函数中,程序使用一个多分支语句(Switch-case),根据标志将执行流程转移到相应的例程函数。主函数的参数 Flag 标志值是由系统(Simulink 引擎)调用时给出的。读者可以打开并阅读该 M 文件 S-函数模板文件。

(1) 打开模板文件的方法有两种,用户可以在 MATLAB 命令窗口中键入:

```
>> edit sfuntmpl
```

或者双击 User-defined Function \S-function Examples\M-file S-functions\Leveal-1 M-file S-functions1\ Leveal-1 M-file template 模块。

(2) M 文件 S-函数模板文件代码。M 文件 S-函数模板文件的代码如下:

```
%主函数。
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
switch flag,
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;
  case 1,
```

```
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u);
  case 3,
    sys=mdlOutputs(t,x,u);
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9,
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end                          % 主函数结束，下面是各个子函数，即各个仿真例程。

        %  初始化例程子函数：提供状态、输入、输出、采样时间数目和初始状态的值。
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;                  %  生成 sizes 数据结构
sizes.NumContStates   = 0;         %  连续状态个数，缺省为 0。
sizes.NumDiscStates   = 0;         %  离散状态个数，缺省为 0。
sizes.NumOutputs      = 0;         %  输出量个数，缺省为 0。
sizes.NumInputs       = 0;         %  输入量个数，缺省为 0。
sizes.DirFeedthrough = 1;          %  有无直接馈入，有取 1，无取 0，缺省为 1。
sizes.NumSampleTimes = 1;          %  采样时间个数，至少取 1。
sys = simsizes(sizes);             %  返回 sizes 数据结构所包含的信息。
x0   = [];                         %  设置初始状态。
str = [];                          %  保留变量，置为空矩阵。
ts   = [0 0];    %  采样时间：[采样周期  偏移量]，采样时间取 0 表示为连续系统。

    % 计算导数例程子函数：计算连续状态的导数，用户需在此例程输入连续状态方程。
    % 该子函数可以不存在。
function sys=mdlDerivatives(t,x,u)
sys = [];                          %  sys 表示连续状态导数。

    % 状态更新例程子函数：计算离散状态的更新。
    % 用户除了需在此输入离散状态方程外，还可以输入其他每个仿真步长都有必要执
      行的代码。
    % 该子函数可以不存在。
function sys=mdlUpdate(t,x,u)
sys = [];                          %  sys 表示下一个离散状态，即 x(k+1)。
```

```
    % 计算输出例程子函数：计算模块输出。该子函数必须存在，用户在此输入系统的
      输出方程。
function sys=mdlOutputs(t,x,u)
sys = [];                             % sys 表示系统输出 y。

                        % 计算下一个采样时间，只有变采样时间系统才调用此仿真例程。
function sys=mdlGetTimeOfNextVarHit(t,x,u)
sampleTime = 1;                       % 设置下一次的采样时间是 1s 以后。
sys = t + sampleTime;                 % sys 表示下一个采样时间点。

                 % 仿真结束调用的例程函数：用户需在此输入结束仿真所需要的必要工作。
function sys=mdlTerminate(t,x,u)
sys = [];
```

(3) M 文件 S-函数模板文件的几点说明。

主函数包含 4 个输出参数：sys 数组返回某个子函数，它的含义随着调用子函数的不同而不同；x0 为所有状态的初始化向量；str 是保留参数，总是一个空矩阵；ts 返回系统采样时间。

主函数的 4 个输入参数分别是采样时间 t，状态 x，输入 u 和仿真流程控制标志变量 flag。输入参数后面还可以附加一系列用户仿真需要的参数。

编写用户自己的 S-函数时，应将函数名 sfuntmpl 改为 S-function 模块中设置的函数名。

读者可能已经发现一个令人困惑的问题：不论在哪个仿真阶段，例程子函数的返回变量都是 sys。要搞清楚这个问题，还要回到 Simulink 如何调用 S-函数上来。前面讲过，Simulink 在每个仿真步长的仿真循环中的每个仿真阶段都要调用 S-函数。在调用时，Simulink 不但根据所处的仿真阶段为 flag 传入不同的值，还会为返回变量 sys 指定不同的角色。即是说尽管是相同的 sys 变量，但在不同的仿真阶段其意义是不相同的，这种变化由 Simulink 自动完成。

10.3.2 M 文件 S-函数的应用举例

了解了 M 文件 S-函数模板文件的代码、代码中各个部分完成的功能及各参数的含义后，用户可以着手利用 S-函数进行系统仿真了。下面使用 M 文件 S-函数实现几种不同的系统。

1. 含用户参数的简单系统

M 文件 S-函数除了模板文件中要求的几个必须的参数外，还可以加入用户自定义的参数，自定义参数需要在 S-函数的输入参数中列出。在含用户自定义参数的 S-函数中，主函数要做适当的修改以便将自定义参数传递到子函数中，子函数也需要相应的修改以便接受自定义参数。在编写 S-函数时，应能区分哪些参数会影响哪一个子函数的执行；要针对这些参数做相应的修改。还需注意的一点是，S-function 模块中的参数设置对话框中的参数输入顺序应与 S-函数中自定义参数的顺序相同。

例 10.2 用 S-函数实现 gain 模块：增益值作为 S-函数用户自定义参数输入。

解 (1)编写 S-函数的源文件。修改 M 文件 S-函数的主函数：增加自定义参数，采用新

的函数名：

```
function [sys,x0,str,ts] = sfun_var gain(t,x,u,flag,gain)
```

由于增益参数只是用来计算输出值的，因而对初始化例程和计算输出例程子函数做修改，其他例程均不需调用，不用做修改。

```
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(gain);
case 3,
    sys=mdlOutputs(t,x,u,gain);
```

修改初始化例程子函数：

```
function [sys,x0,str,ts]=mdlInitializeSizes(gain)
sizes. NumContStates    = 0;
sizes. NumDiscStates              = 0;
sizes. NumOutputs                 = 1;
sizes. NumInputs                  = 1;
sizes. DirFeedthrough             = 1;
```

定义计算输出例程子函数

```
function sys=mdlOutputs(t,x,u,gain)
sys = gain * u;          % 输出=增益×输入。
```

(2)建立如图10.6所示的系统仿真模型，将自定义参数设置为3，运行仿真，仿真结果如图10.6所示，验证了S-函数的正确性。

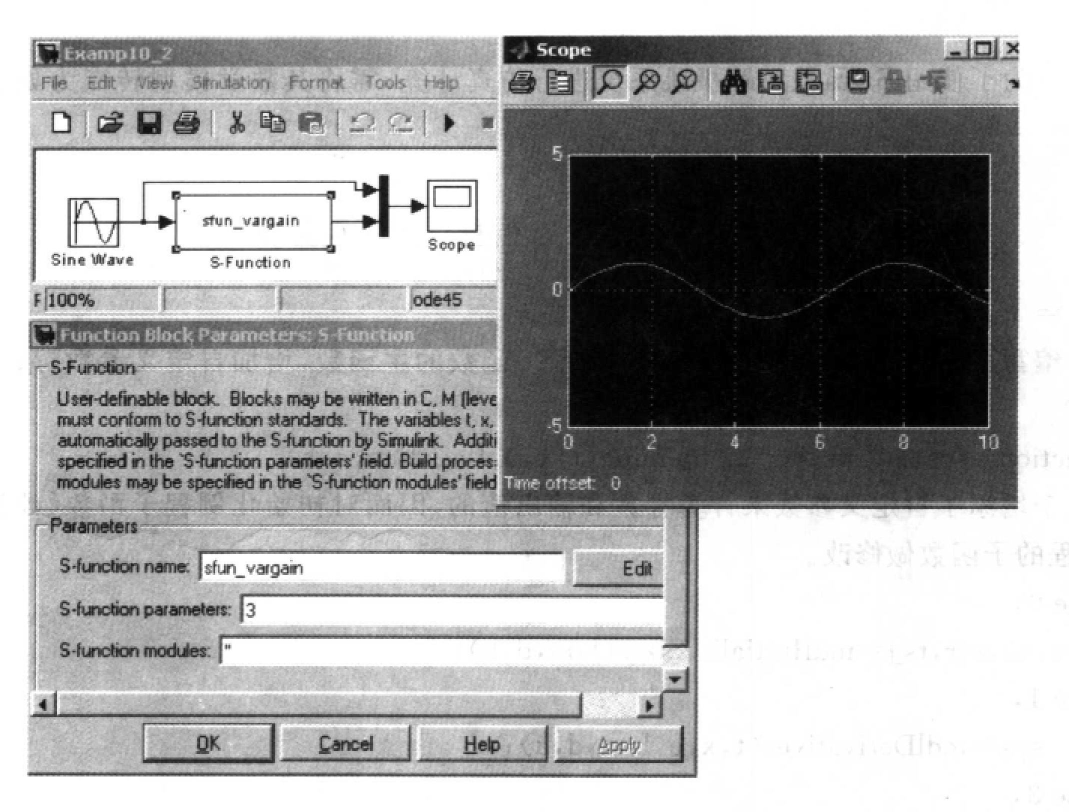


图10.6　含用户参数的简单系统仿真

2. 连续系统的 S-函数描述

用 S-函数实现一个连续系统时，首先需要修改初始化例程子函数 mdlInitialSizes，包括正确设置连续状态个数、状态初始值和采样时间；其次，需要编写计算导数例程子函数，将状态的导数向量通过 sys 变量返回。如果存在多个系统状态，可以通过索引 $x(1)$，$x(2)$ 等得到各个状态，此时，变量 sys 即为一个向量，包含所有各个连续状态的导数。最后，也需在计算输出例程子函数中编写系统的输出方程。下面举例说明。

例 10.3　试用 S-函数对蹦极跳系统做仿真分析。

蹦极跳是一种挑战身体极限的运动，蹦极者系着一根弹力绳从高处的桥梁或山崖向下跳。如果蹦极者系在一个弹性系数为 k 的弹力绳索上。定义绳索下端的初始位置为 0，则蹦极者受到的弹性力是 $b(x)=\begin{cases}-kx & x>0\\ 0 & x\leqslant 0\end{cases}$，整个蹦极跳系统的数学模型为

$$m\ddot{x}=mg+b(x)-a_1\dot{x}-a_2|\dot{x}|\dot{x}$$

其中，m 为物体的质量，g 为重力加速度，x 为物体的位置，第二项是物体受到的弹性力，第三与第四项表示空气的阻力。

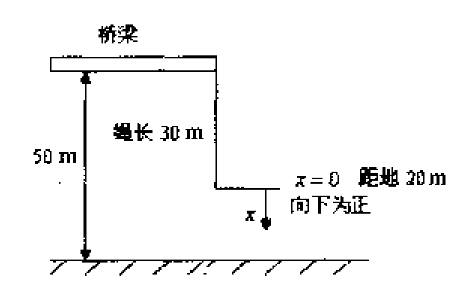


图 10.7　蹦极跳系统坐标系

设桥梁距离地面高度为 50 m，绳索长度 30 m，由此可知蹦极者站在桥上时的初始位置 $x(0)=-30$，初始速度 $\dot{x}(0)=0$；根据对蹦极跳系统的描述，可以分析出该系统的坐标系如图 10.7 所示。其余参数分别取为 $k=50$，$a_1=a_2=1$，$m=70$ kg，$g=10$ m/s^2。下面我们分析此蹦极跳系统对于质量为 70 kg 的蹦极者是否安全。

解　为了使用 S-函数对蹦极跳系统进行仿真计算，需将其数学模型转变为如下状态方程形式

$$\begin{cases}\dot{x}_1=x_2\\ \dot{x}_2=g+\dfrac{b(x_1)}{m}-\dfrac{a_1x_2}{m}-\dfrac{a_2|x_2|x_2}{m}\end{cases}$$

其中 $x_1=x$，$x_2=\dot{x}$

(1)编写 S-函数的源文件：修改 M 文件 S-函数的主函数，增加自定义参数，采用新的函数名：

```
function [sys,x0,str,ts] = jumping(t,x,u,flag,l,m,d,k)
```

由于增添了自定义参数来计算导数和输出值的，因而对初始化例程子函数、计算导数和输出例程的子函数做修改。

```
case 0,
[sys,x0,str,ts]=mdlInitializeSizes(l,m,d,k);
case 1,
    sys=mdlDerivatives(t,x,u,l,m,d,k);
case 3,
    sys=mdlOutputs(t,x,u,l,m,d,k);
```

修改初始化例程子函数：

```
function [sys,x0,str,ts]=mdlInitializeSizes(l,m,d,k)
sizes.NumContStates    = 2;
sizes.NumDiscStates    = 0;
sizes.NumOutputs       = 1;
sizes.NumInputs        = 0;
sizes.DirFeedthrough   = 1;
x0   = [-1;0];
定义计算导数例程子函数
function sys=mdlDerivatives(t,x,u,l,m,d,k)
if x(1)>=0
    b=-k*x(1);
else
    b=0;
end
sys = [x(2);10+b/m-1/m*x(2)-1/m*abs(x(2))*x(2)];
定义计算输出例程子函数
function sys=mdlOutputs(t,x,u,l,m,d,k)
sys = [d-l-x(1)];
```

(2)建立如图10.8所示的系统仿真模型,并按正确的顺序输入自定义参数l, m, d, k的值,运行仿真,仿真结果如图10.8所示。由仿真曲线可知,本蹦极跳系统对于质量70 kg的蹦极者来说是非常危险的,因为仿真结果显示蹦极者有触地的危险,为了满足大体重的蹦极爱好者的要求,必须对系统参数做适当的调整。具体如何调整,将在第十一章再做介绍。

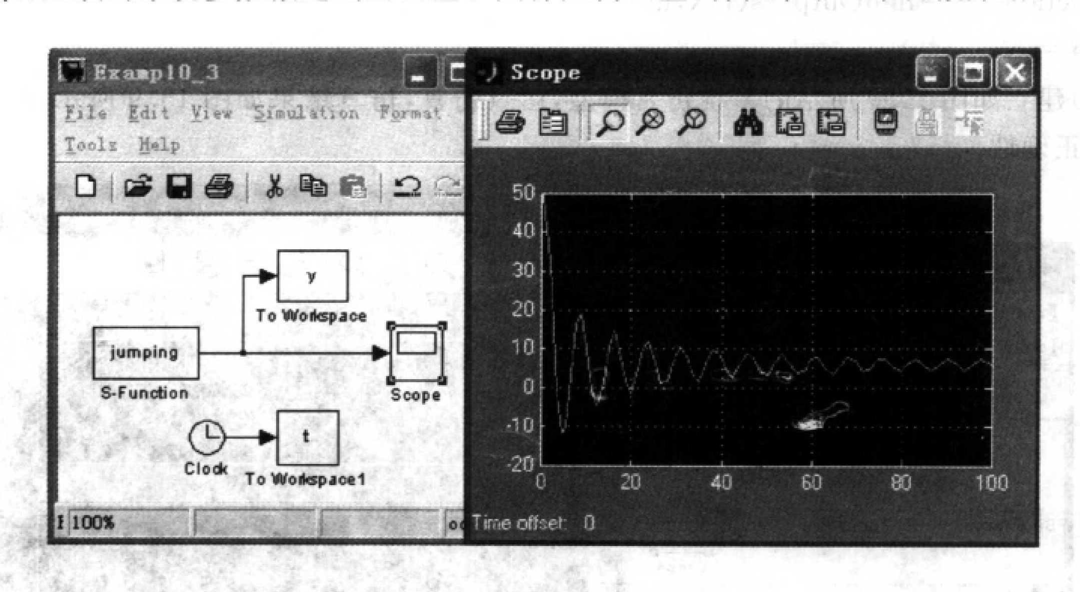


图10.8　蹦极跳系统的仿真模型及仿真结果

3. 离散系统的S-函数描述

用S-函数实现离散系统时,首先要对初始化例程子函数mdlInitializeSizes做适当修改,包括声明离散状态的个数、对状态进行初始化、确定采样时间等,其次需要定义状态更新和计

算输出例程子函数 mdlUpdate 和 mdlOutputs，分别需要输入表示离散系统的离散状态方程和输出方程。

例 10.4 编写 S-函数实现输出对输入的单位延迟，即 $y(k+1)=u(k)$。

解 单位延迟系统的状态方程可以表示为

$$x(k+1)=u(k),\quad y(k)=x(k)$$

(1)编写 S-函数的源文件。

1) 修改 M 文件 S-函数的主函数：采用新的函数名：

```
function [sys,x0,str,ts] = sfun_unitdelay(t,x,u,flag)
```

2) 修改初始化例程子函数：

```
sizes.NumContStates    = 0;
sizes.NumDiscStates    = 1;
sizes.NumOutputs       = 1;
sizes.NumInputs        = 1;
sizes.DirFeedthrough   = 0;
sizes.NumSampleTimes = 1;
x0    = 0;
ts    = [0.1 0];
```

定义状态更新例程子函数：

```
function sys=mdlUpdate(t,x,u)
sys = u;
```

定义计算输出例程子函数：

```
function sys=mdlOutputs(t,x,u)
sys = x;
```

(2)建立如图 10.9 所示的系统仿真模型，运行仿真，仿真结果如图 10.9 所示，验证了 S-函数的正确性。

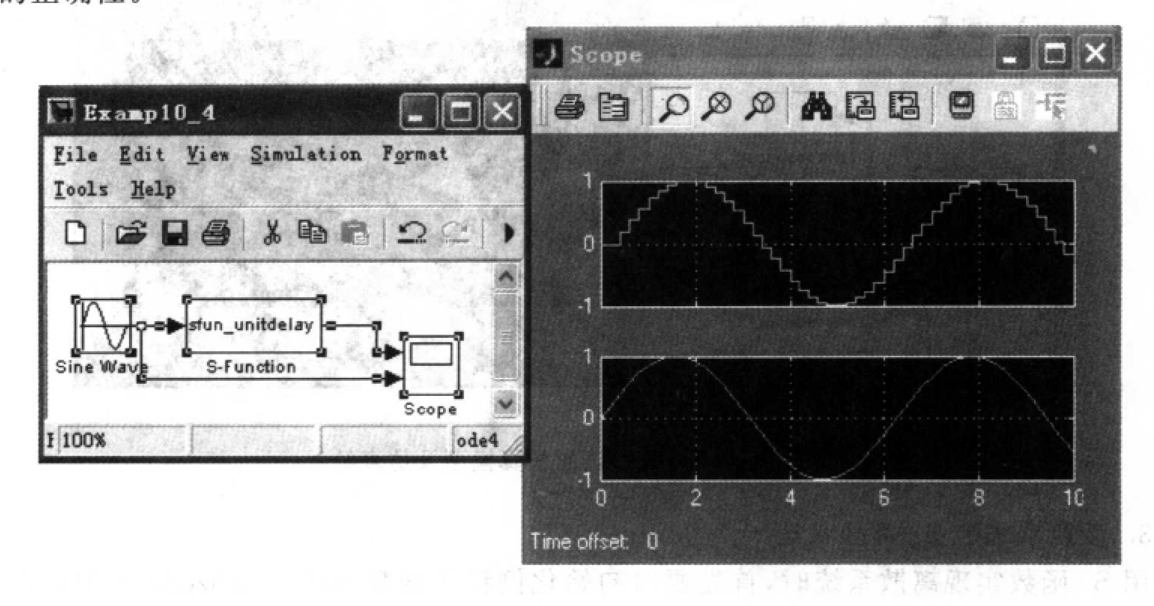


图 10.9 单位延迟系统的仿真模型及结果

4. 混合系统的 S-函数描述

既包含离散状态，又包含连续状态的系统称为混合系统。这里介绍如何用 S-函数描述一个连续积分器后接一个离散单位延迟的混合系统。

例 10.5　编写 S-函数实现一个连续积分器后接一个离散单位延迟的混合系统。

解　首先，在初始化例程要定义两个采样时间：连续采样时间和离散采样周期，于是在这两个采样时间决定的采样点，Simulink 都会调用 S-函数源文件，并依次用 flag 来指定要调用的例程。

在混合系统中，与离散状态和连续状态有关的 S-函数例程都有可能执行，但各自的执行时间不同。连续状态的导数计算是在每个微时间步都要进行的，但离散状态的更新以及系统的输出计算，只在离散采样点到达时才完成。但这一点 Simulink 是无法区分什么时候该更新什么时候不该更新。因为，Simulink 从初始化获得的信息只是该系统既有离散状态又有连续状态，据此只能决定该函数的仿真循环应该包含连续状态的微分计算和离散状态的更新例程，没有足够的信息来判断当前时刻是否是离散采样点。

这时常用的技巧是，在 mdlUpdate 和 mdlOutputs 中由程序判断当前时刻是否是离散采样点。mdlUpdate 例程中完成此功能的语句是：

```
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);       % 是离散采样点，将离散状态更新为当前连续状态。
else
  sys = [];         % 不是离散采样点，离散状态保持不变。
end
```

mdlOutputs 例程中完成此功能的语句类似，这里就不赘述了。

本系统的 S-函数源文件的代码如下：

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
dperiod = 1;            % 设置离散采样周期和偏移量。
doffset = 0;

switch flag
  case 0
    [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset);
  case 1
    sys=mdlDerivatives(t,x,u);
  case 2,
    sys=mdlUpdate(t,x,u,dperiod,doffset);
  case 3
    sys=mdlOutputs(t,x,u,doffset,dperiod);
   case 9
    sys = [];          % do nothing
  otherwise
```

```
    error(['unhandled flag = ',num2str(flag)]);
  end

function [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset)

sizes = simsizes;
sizes.NumContStates  = 1;      % 一个连续状态。
sizes.NumDiscStates  = 1;      % 一个离散状态。
sizes.NumOutputs     = 1;      % 一个输出。
sizes.NumInputs      = 1;      % 一个输入。
sizes.DirFeedthrough = 0;      % 没有直接馈入。
sizes.NumSampleTimes = 2;      % 两个采样时间。
sys = simsizes(sizes);
x0   = ones(2,1);              % 初值均设置为 1。
str = [];
ts   = [0 0;                   % 采样时间是[0 0]表示连续系统。
        dperiod doffset];      % 离散采样时间及偏移量,主程序开始时已设置。
function sys=mdlDerivatives(t,x,u)
sys = u;                       % 连续系统是积分环节。
function sys=mdlUpdate(t,x,u,dperiod,doffset)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);                  % 离散系统是单位延迟。
else
  sys = [];
end

function sys=mdlOutputs(t,x,u,doffset,dperiod)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(2);                  % 输出采样延迟。
else
  sys = [];
end
```

10.4 编写 C MEX S-函数

第 10.3 节介绍的 M 文件 S-函数由于具有易于编写和理解的特点,在仿真计算中得到了广泛的应用。但是,它有一些缺点:首先,M 文件 S-函数使得每个仿真步都必须激活 MAT-

LAB 解释器，以致仿真速度变慢；其次，当需要利用 RTW 从 Simulink 框图生成实时代码时，框图中不能含有 M 文件 S-函数。而 C MEX S-函数不仅运算速度快，而且可以用来生成独立的仿真程序。现在的 C 语言编写的程序还可以方便地通过包装程序结合至 C MEX S-函数中。C MEX S-函数结合了 C 语言的优势，可以实现对操作系统和硬件的访问，实现与串口或网络的通信，编写设备驱动程序等。

本节介绍 C MEX S-函数的概念及编程使用方法。

10.4.1　MEX 文件

M 文件 S-函数在 MATLAB 环境下可以通过解释器直接执行，而 C 文件或其他语言编写的 S-函数，则需要先编译成可以在 MATLAB 内运行的二进制代码：动态链接库或静态链接库，然后才可以使用，这些经过编译的二进制文件就称作 MEX 文件。在 Windows 系统下 MEX 文件后缀是.dll。要将 C 文件 S-函数编译成动态链接库，需 MATLAB 命令窗口键入：

```
>> mex my_sfunction.c        % my_sfunction.c 是用户自己编写的 C 文件 S-函数文件名。
```

要使用 MEX 命令，需要先在系统中安装 C 编译器。如果系统中还没有设置编译器，则要在命令窗口键入：

```
>> mex-setup
```

然后按照提示选取 VC，BC 或其他 C 编译器。建议使用 VC 编译器。生成的文件 my_sfunction.dll 即是所需的动态链接库。当 C 文件中使用了其他库文件时，编译时应该在其后面加上引用的库文件名。如

```
>> mex my_sfunction.c kernel32.lib
```

其实 M 文件 S-函数也可以编译成 MEX 文件再使用，此时 MATLAB 调用的是动态链接库而不是 M 文件本身。M 文件 S-函数编译成 MEX 文件实际上是先用特定的命令将其转换成 C 文件 S-函数，再调用 C 编译器编译链接的。这样编译的代码一般比 M 文件的 S-函数执行速度快，且其代码是二进制的，具有保密性。要将 M 文件编译成 MEX 文件，可在 MATLAB 命令窗口键入：

```
>> mcc -S mfilename
```

其中，-S 选项表示编译的是 S-函数。在 Windows 系统下生成文件 mfilename.c，mfilename.h，mfilename_simulink.c 和 mfilename.dll。在这个过程中，mcc 自动调用了 mex 命令将生成的 C 代码编译成动态链接库。

10.4.2　C MEX S-函数模板

与 M 文件 S-函数类似，Simulink 也为用户提供了编写 C MEX S-函数的模板。常用模板文件是 sfuntmpl_basic.c。该模板为用户提供了 C MEX S-函数的框架结构。文件中包含了常用的几个例程，这对于一般的应用已经足够了。文件 sfuntmpl_doc.c 则包含了所有的例程，并附有详细的说明。

每个 C MEX S-函数的开头应包含下列语句：

```
#define S_FUNCTION_NAME    your_sfunction_name_here
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
```

其中 your_sfunction_name_here 是用户要编写的 S-函数的名字，即 S-function 模块要输入的 S-函数名。S-函数的格式随之 Simulink 版本的不同而略有不同，这里 S_FUNCTION_LEVEL 说明了该 S-函数适用的 Simulink 版本。头文件 simstruc.h 定义了 S-函数的一个重要数据结构。S-函数的有关信息都保存在此数据结构中。另外。头文件 simstruc.h 还包含着其他重要的头文件，如 tmwtypes.h。头文件 tmwtypes.h 定义了各种数据类型。

另外，和编写普通的 C 文件相同，在文件的顶部还应包含适当的头文件或定义其他宏或者变量。在它的尾部必然包含下面代码：

```
#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

宏 MATLAB_MEX_FILE 用于告诉编译器该 S-函数正被编译成 MEX 文件。这组代码的含义是如果该文件正被编译成 MEX 文件(Windows 下为 dll 文件)，包含 simulink.c；如果正在使用 RTW 将整个 Simulink 框图编译成实时的独立程序(Windows 下为 exe 文件)，包含头文件 cg_sfun.h。模板中其他的代码是几个和 M 文件 S-函数中功能类似的 S-函数例程。和 M 文件 S-函数不同，Simulink 不是通过显式的 flag 参数来指定调用 C MEX S-函数例程的。这是因为 Simulink 在交互时会自动地在合适的时候调用每个 S-函数的例程。下面分别介绍 C MEX S-函数在不同仿真阶段的例程。

1. 初始化

C MEX S-函数的初始化部分包括下面 3 个不同的例程函数：

(1) mdlInitializeSizes：在该函数中给出各种数量信息；

(2) mdlInitializeSampleTimes：在该函数给出采样时间；

(3) mdlInitializeConditions：在该函数给出初始状态。

mdlInitializeSizes 通过宏函数对状态、输入、输出等进行设置。工作向量的维数也是在此例程中设置。C MEX S-函数还需要在此例程声明期望的输入参数的个数。表 10.2 列出了初始化所用到的部分宏函数。初始化工作实际上都是通过宏函数访问 SimStruct 数据结构的。

2. 使用输入和输出

在 C MEX S-函数中，对输入输出的操作也是通过描述该 S-函数的 SimStruct 进行的。比如，在 C MEX S-函数中如果需要对输入进行操作，需要先使用宏函数：

```
input=ssGetInputPortRealSignalPtrs(S,index)
```

该宏函数的返回中含有指向输入向量的指针，其中每个元素通过 *input[i]来访问。同样，如果需要对输出进行操作，需要先使用下列宏函数得到指向输出的指针：

```
output=ssGetOutputPortRealSignal (S,index)
```

有关输入与输出相关的宏函数见表 10.3。

表 10.2　S-函数初始化所用的宏函数

<table>
<tr><th>宏函数定义</th><th colspan="2">功能描述</th></tr>
<tr><td>ssSetNumContStatus(S, numContStates)</td><td colspan="2">设置连续状态个数</td></tr>
<tr><td>ssSetNumDiscStatus(S, numDiscStates)</td><td colspan="2">设置离散状态个数</td></tr>
<tr><td>ssSetNumOutputs(S, numOutputs)</td><td colspan="2">设置输出个数</td></tr>
<tr><td>ssSetNumInputs(S, numInputs)</td><td colspan="2">设置输入个数</td></tr>
<tr><td>ssSetDirectFeedthrough (S,dirFeedThrou)</td><td colspan="2">设置是否存在直接前馈</td></tr>
<tr><td>ssSetNum SampleTimes (S, numSampleTimes)</td><td colspan="2">设置采样时间数目</td></tr>
<tr><td>ssSetNumInputArgs(S, numInputArgs)</td><td colspan="2">设置输入参数个数</td></tr>
<tr><td>ssSetNumIWork (S, numIWork)</td><td>设置整数型工作向量维数</td><td rowspan="3">实际上是为各个工作向量分配内存提供依据</td></tr>
<tr><td>ssSetNumRWork (S, numRWork)</td><td>设置实数型工作向量维数</td></tr>
<tr><td>ssSetNumPWork (S, numPWork)</td><td>设置指针型工作向量维数</td></tr>
</table>

表 10.3　有关输入与输出的宏函数

宏函数	功能描述
ssGetInputPortRealSignalPtrs	获得指向输入的指针(double 型)
ssGetInputPortSignalPtrs	获得指向输入的指针(其他数据类型)
ssGetInputPorWidth	获得输入信号的宽度
ssGetInputPortOffsetTime	获得输入端口的采样时间偏移量
ssGetInputPortSampleTime	获得输入端口的采样时间
ssGetOutputPortRealSignal	获得指向输出的指针
ssGettOutputPorWidth	获得输出信号的宽度
ssGetOutputPortOffsetTime	获得输出端口的采样时间偏移量
ssGetOutputPortSampleTime	获得输出端口的采样时间

3. 使用参数

使用用户自定义参数时,在初始化时,必须说明参数的个数。为了得到指向存储参数的数据结构的指针,必须使用宏函数 ptr=ssGetSFcnParam(S,index);为了得到存储在此数据结构中的指向参数值本身的指针,需要使用宏:mxGetPr(ptr);使用参数值时要使用宏:param_value= * mxGetPr(ptr)。

4. 使用状态

若S-函数中包含连续或离散状态,则需要编写 mdlDerivatives 或 mdlUpdate 子函数。使用宏 ssGetRealDiscStates(S) 可以得到指向离散状态向量的指针;使用宏函数 ssGetContStates(S) 可以得到指向离散状态向量的指针。在 mdlDerivatives 函数中,连续状态的导数由状态和输入量计算而得,可以通过宏 * dx=ssGetX(S)将 SimStruct 结构体中的连续状态导数指针指向得到的结果,然后修改所指向的值。在多状态情况下,需要通过索引得到 dx 中的每

个元素。它们被返回给求解器，求解器再积分求出状态。需要指出的是，在离散系统中，没有对应于 dx 的变量，由于状态是由 S-函数更新的，不需要求解器再做工作。

10.4.3 C MEX S-函数应用举例

这里给出几个 C MEX S-函数的示例，以帮助用户更快地掌握 C MEX S-函数的编程方法。

例 10.6 S-函数 csfunc 描述了一个用状态方程表示的线性连续系统：

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

下面是其 C MEX S-函数的完整源代码和注释。

```
#define S_FUNCTION_NAME csfunc1                              /*  S-函数名称   */
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#define U(element) ( * uPtrs[element])            /* 宏定义，方便对输入的索引 */
                                        /* 定义状态方程中的参数 ABCD 阵   */
static real_T A[2][2]={ { -0.09, -0.01 } ,
                        {  1    ,  0     }
                                              /* 在 C S 函数中有一套自己的数据 */
                      };                   /* 类型表示方法，real_T 表示双精  */
static real_T B[2][2]={ {  1    , -7     } ,
                        {  0    , -2     }
                      };
static real_T C[2][2]={ {  0    , 2      } ,
                        {  1    , -5     }
                      };
static real_T D[2][2]={ { -3    , 0      } ,            /* 存在直接馈入 */
                        {  1    , 0      }
                      };
static void mdlInitializeSizes(SimStruct * S)
{
    ssSetNumSFcnParams(S, 0);                   /* 不含用户参数，设置为零 */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* 若参数不匹配，Simulink 将发出警告 */
    }
    ssSetNumContStates(S, 2);                  /* 系统有两个连续状态 */
    ssSetNumDiscStates(S, 0);                /* 系统无离散状态 */
    if (!ssSetNumInputPorts(S, 1)) return;  /* 如果输入端口数不为 1，则返回 */
    /* S-functions 块只有一个输入端口，当需要多个输入时，必须使用 mux 模块把需
要输入的信号合成一个向量 */
```

```
    ssSetInputPortWidth(S, 0, 2);    /* 输入信号宽度为 2 */
    ssSetInputPortDirectFeedThrough(S, 0, 1);    /* 设置馈通标志为 1 */
    if (! ssSetNumOutputPorts(S, 1)) return;  /* 如果输出端口数不为 1,则返回 */
    ssSetOutputPortWidth(S, 0, 2);    /* 输出信号宽度为 2 */
    ssSetNumSampleTimes(S, 1);    /* 1 个采样时间 */
    ssSetNumRWork(S, 0);          /* 不使用工作向量 */
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    /* 连续系统的采样时间设置为 0,等同于 ssSetSampleTime(S, 0, 0)  */
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetContStates(S);    /* 获得指向连续状态的指针 */
    int_T  lp;
    for (lp=0;lp<2;lp++) {
        *x0++=0.0;                  /* 各状态初值设置为 0  */
    }
}
static void mdlOutputs(SimStruct *S, int_T tid)
{   /* 获得指向输出向量、连续状态向量和输入端口的指针  */
    real_T          *y    = ssGetOutputPortRealSignal(S,0);
    real_T          *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
     UNUSED_ARG(tid); /* not used in single tasking mode */
                    /* y=Cx+Du              输出方程 */
    y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
    y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
}
  #define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
```

```
{
    real_T        * dx    = ssGetdX(S);  /* 获得指向状态导数向量的指针  */
    real_T        * x     = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    /* xdot=Ax+Bu    状态方程   */
    dx[0]=A[0][0] * x[0]+A[0][1] * x[1]+B[0][0] * U(0)+B[0][1] * U(1);
    dx[1]=A[1][0] * x[0]+A[1][1] * x[1]+B[1][0] * U(0)+B[1][1] * U(1);
}
static void mdlTerminate(SimStruct * S)
{
    UNUSED_ARG(S); /* unused input argument */
}
#ifdef  MATLAB_MEX_FILE     /* 是否编译成 MEX 文件？ */
#include "simulink.c"       /* 包含 MEX 文件的接口机制 */
#else
#include "cg_sfun.h"        /* 代码生成注册函数  */
#endif
```

例 10.7　由 C MEX S-函数实现对下列 van der pole 方程的求解。

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -m(x_1^2 - 1)x_2 - x_1$$

并要求 m 值和状态初值 x_0 由用户输入。

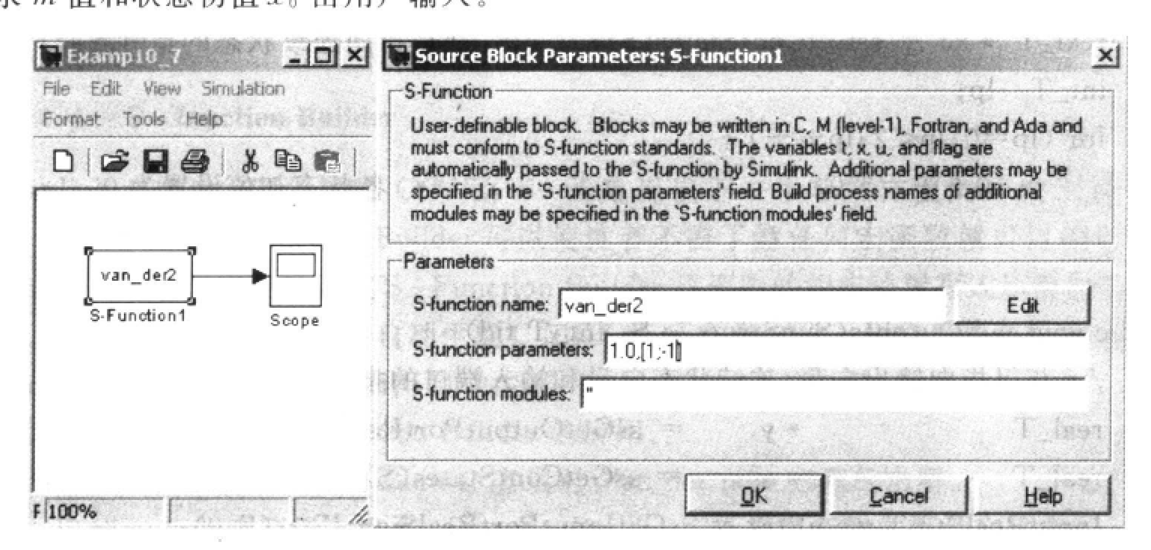


图 10.10　例 10.7 系统仿真模型及 S-函数参数对话框

解　建立如图 10.10 的系统仿真模型。Van der pole 方程由 C MEX S-函数实现，其 C 程序代码如下：

```
#define S_FUNCTION_NAME   van_der2
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
```

```
#define M ssGetSFcnParam(S,0)
#define X00 ssGetSFcnParam(S,1)
static void mdlInitializeSizes(SimStruct *S)
                                        /* 得到指向存储参数的数据结构的指针 */
{
    ssSetNumSFcnParams(S, 2);                 /* 两个用户参数 */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return;          /* Parameter mismatch will be reported by Simulink */
    }
    ssSetNumContStates(S, 2);                 /* 两个连续状态 */
    ssSetNumDiscStates(S, 0);
    if (!ssSetNumInputPorts(S, 0)) return;     /* 系统无输入端口 */
    if (!ssSetNumOutputPorts(S, 1)) return;
      ssSetOutputPortWidth(S, 0, 2);          /* 输出信号为2维 */
    ssSetNumSampleTimes(S, 1);                /* 一个采样时间 */
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    real_T *x0 = ssGetContStates(S);
 const real_T    *x00=mxGetPr(X00);
                                /* 得到存储在数据结构中的指向参数值本身的指针 */
    x0[0] =x00[0];
    x0[1] = x00[1];
}
static void mdlOutputs(SimStruct *S, int_T tid)
{
    real_T          *y = ssGetOutputPortRealSignal(S,0);
    real_T          *x = ssGetContStates(S);
    y[0]=x[0];
```

```
    y[1]=x[1];
    }
#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
{
    real_T *dx     = ssGetdX(S);
    real_T *x      = ssGetContStates(S);
        const real_T      m = *mxGetPr(M);
    dx[0] = -m*x[0] * (1.0 - x[1] * x[1]) - x[1];
    dx[1] = x[0];
}
static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S);          /* unused input argument */
}
#ifdef  MATLAB_MEX_FILE      /* Is this file being compiled as a MEX-file? */
#include "simulink.c"        /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"         /* Code generation registration function */
#endif
```

由例 10.6 和例 10.7 给出的程序可以看出，C MEX S-函数是通过一套宏函数获得指向存储在 SimStruct 中的输入、输出、状态、状态导数及用户参数的指针来引用这些变量，从而完成对系统的描述。

10.4.4 S-function Builder

Simulink 为用户编写常用的 C MEX S-函数提供了一个非常方便的开发工具——S-Function Builder。S-Function Builder 可以使读者不需了解任何宏函数就可以编写出自己的 C MEX S-函数。用户只需在 S-Function Builder 界面中的相应位置写入所需的信息和代码即可。S-Function Builder 会自动生成 C MEX S-函数源文件。用户只要单击 Build 按钮，S-Function Builder 就会自动编译，自动生成用户所需的 MEX 文件。

下面通过例子介绍 S-Function Builder 的应用。

例 10.8 利用 S-function Builder 实现对 Van der pole 方程的求解。

解 从 Simulink 浏览器中将 S-function Builder 图标拖至新建的模型文件中。

双击 S-function Builder 图标，即可打开如图 10.11(a)所示的 S-function Builder 界面。用户可以看出 1,4,5,6 选项卡对应着 S-函数的 4 个常用例程。用户只需在相应的例程函数中填写所需的信息和代码即可。下面给出使用 S-function Builder 编写 S-函数的方法。

(1) 在 S-Function name 编辑栏中填写 S-函数名称。

(2) 在图 10.11(a)所示的初始化(Initialization)选项卡中按照提示设置仿真的相关信息。例 10.8 所设置的信息如图 10.11(a)所示。

(3) 在图 10.11(b)所示的数据属性选项卡中设置输入、输出的数据类型、信号维数等信

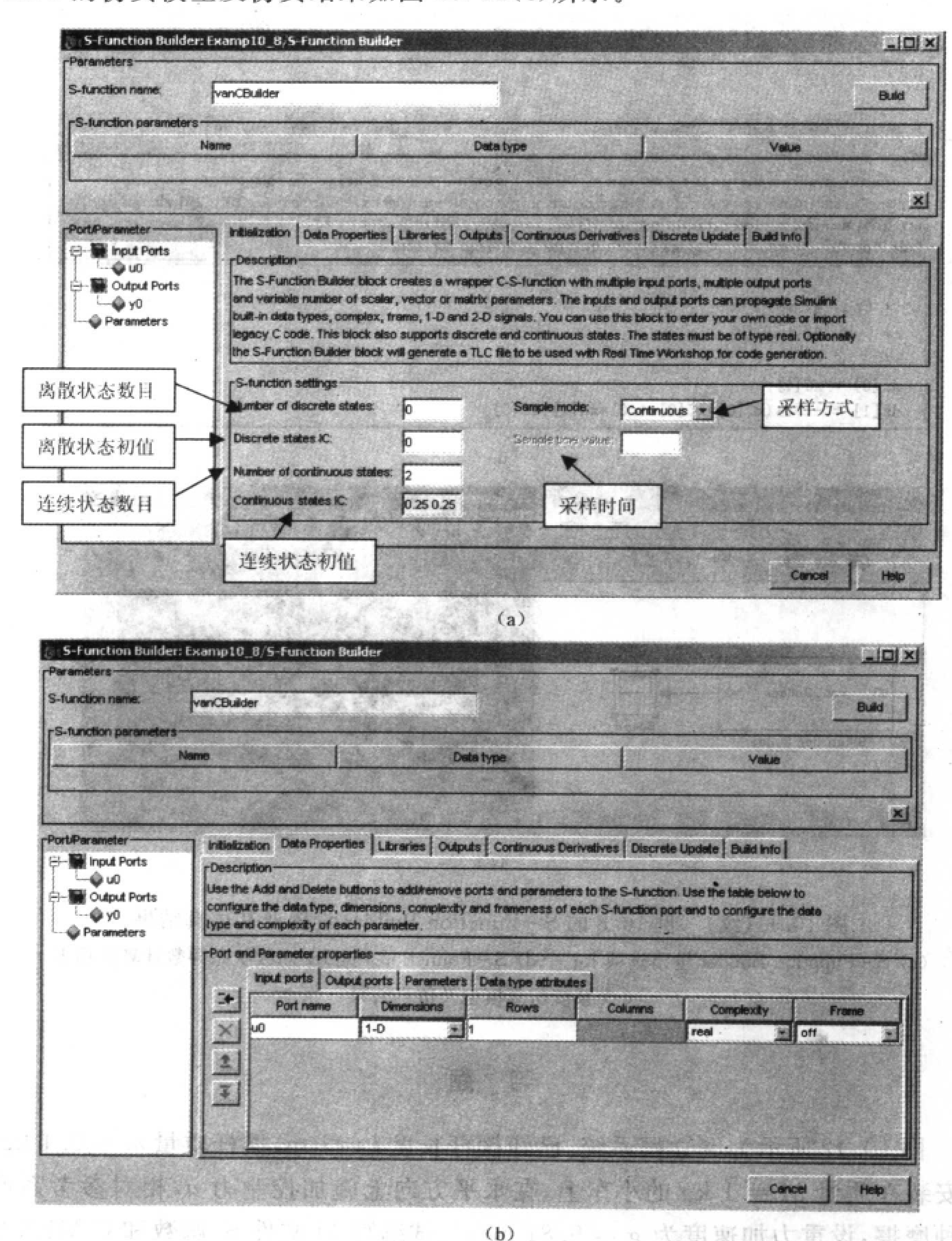
息，设置用户参数的名称、数据类型等。例 10.8 设置如图 10.11(b)所示。

（4）在 Library 选项卡中需要填入所需的库文件（包括目录）、要包含的头文件以及外部函数声明等。

（5）在 Outputs，Continues Derivatives 和 Discrete Update 选项卡中分别填入输出方程、连续状态方程、离散状态方程以及其他用户定制的代码。例 10.8 是连续系统，其 Outputs，Continues Derivatives 选项卡所需填写的代码见图 10.11(c)，(d)所示。

（6）单击 Build 按钮，S－function Builder 会自动生成 C 代码、完成编译链接等工作，并给出编辑链接成功信息。

例 10.8 的仿真模型及仿真结果如图 10.11(e)所示。

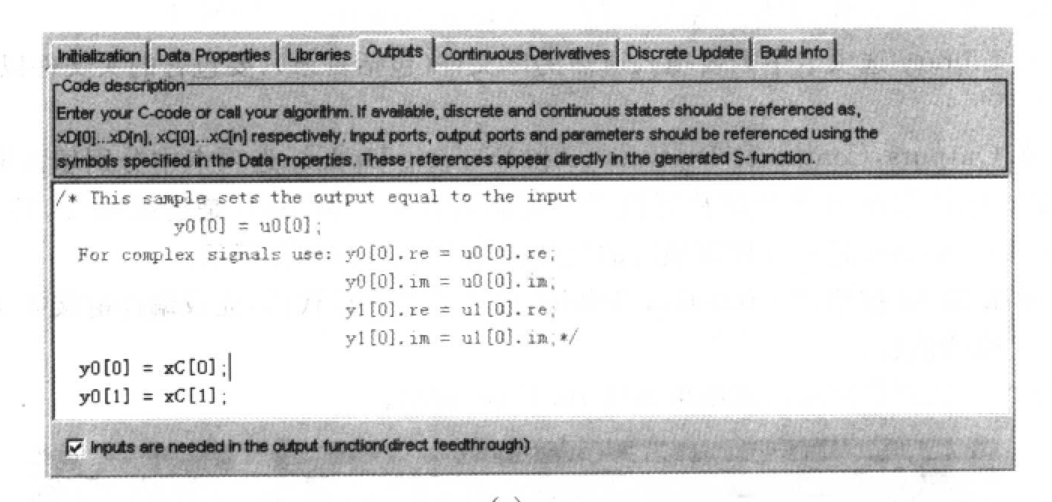


(a)

(b)

图 10.11　例 10.8 的 S－Function Builder 设置及其仿真结果

(a) S－Function Builder 初始化界面；　(b) S－Function Builder 数据属性设置界面

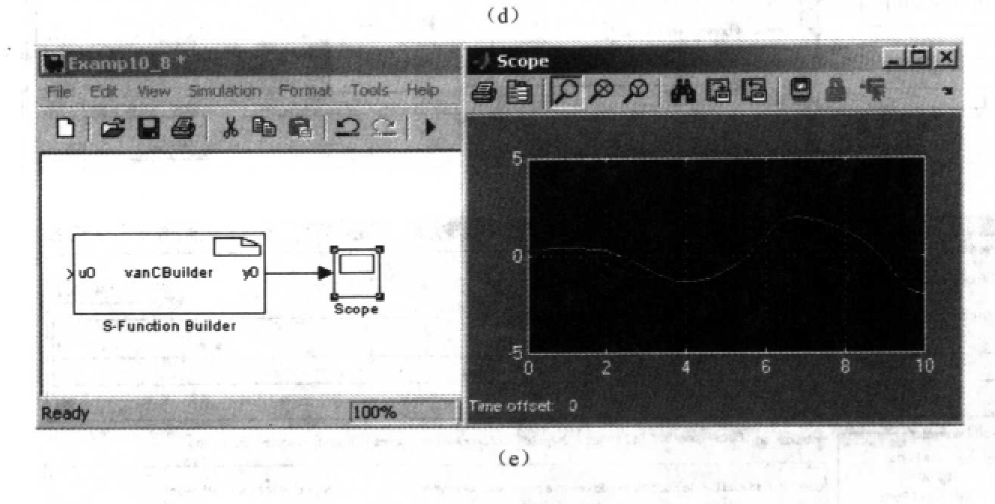


(c)

```
Initialization | Data Properties | Libraries | Outputs | Continuous Derivatives | Discrete Update | Build Info
Code description
This section is optional and use to calculate the derivatives. It is called only if the S-function has one or more continuous
states. The states and derivatives of the S-function are of type double and must be referenced as xC[0], xC[1], etc and dx[0]
and dx[1] etc respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the
Data Properties. These references appear directly in the generated S-function.

/*
  * Code example
  *    dx[0] = xC[0];
 */
dx[0] = xC[1];
dx[1] = -1.0*(xC[0]*xC[0]-1.0)*xC[1]-xC[0];
```

(d)



(e)

图 10.11(续)　例 10.8 的 S-Function Builder 设置及其仿真结果

(c) S-Function Builder 输出选项卡；(d) S-Founction Builder 连续状态导数计算选项卡

(e) 系统仿真模型及仿真结果

习　题

10.1　图 10.12 所示为倒立摆系统。已知摆杆长度 $l = 1$ m，摆杆质量 $m = 0.1$ kg。摆杆底端用铰链安装在质量 $M = 1$ kg 的小车上，在水平方向上施加控制力 u，相对参考系产生位移 z，忽略各种摩擦，设重力加速度为 $g = 9.81\ \mathrm{m/s^2}$。试编写 M 文件 S- 函数和 C MEX S- 函数，建立倒立摆系统的模型并进行仿真。要求用户可以输入系统参数(摆杆长度、摆杆质量、小

车质量）

提示：建立数学模型：小车位置 z，立摆偏离垂直位置的角度是 θ，如图 10.12 所示。

根据力学方程，其运动方程描述如下

$$\begin{cases} \ddot{z} = \dfrac{(u + ml\dot{\theta}^2\sin\theta)\cos\theta - (g\sin\theta + l\dot{\theta}^2\sin\theta\cos\theta)m}{M\cos\theta} \\ \ddot{\theta} = \dfrac{(M+m)(g\sin\theta + l\dot{\theta}^2\sin\theta\cos\theta) - \cos\theta(u + ml\sin\theta)}{Ml\cos^2\theta} \end{cases}$$

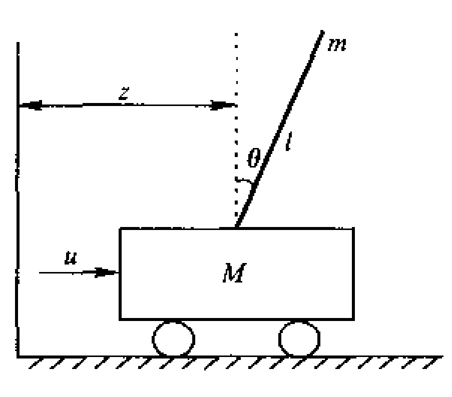


图 10.12　题 10.1 图

第十一章 Simulink 命令行仿真技术及回调函数概念

前面的章节中，动态系统模型的建立、仿真及分析均是使用 Simulink 的图形建模和仿真方式实现的。虽然 Simulink 的图形建模方式能够给用户提供强大的功能与友好的使用界面，使用户可以完成大部分的动态系统的仿真分析工作。但在分析一些系统在不同的参数情况下的性能、在对系统进行调参以满足特定要求或分析系统在不同的输入信号的作用下的响应时，单纯使用 Simulink 的图形建模方式是非常不方便的。

本章将介绍 Simulink 命令行仿真技术。所谓命令行仿真是指在进行动态系统设计、建模、仿真与分析中，使用 MATLAB 命令行的方式对系统的仿真分析进行控制和操作的方法。它允许用户可以从 M 文件来对动态系统进行仿真，这样用户就可以通过 MATLAB 工作空间不断改变系统仿真或模块的参数，循环地运行仿真。

Simulink 命令行仿真技术允许用户使用 M 文件对动态系统进行仿真分析，因而提出了如何在 MATLAB 命令中应用 Simulink 仿真计算的结果，对系统进行更深入的分析以及系统的 Simulink 仿真模型如何使用 MATLAB 的计算的参数等问题。为了使用户能够方便地使用命令行仿真技术，首先介绍 Simulink 与 MATLAB 的接口。

11.1 Simulink 与 MATLAB 的接口

Simulink 是基于 MATLAB 的系统级仿真平台，它与 MATLAB 紧密地集成在一起。Simulink 不仅能够采用 MATLAB 的求解器对动态系统进行求解，还可以与 MATLAB 进行数据交互(从 MATLAB 工作空间中读入数据或向 MATLAB 工作空间中写入数据)。

11.1.1 由 MATLAB 工作空间变量设置系统模块参数

前面章节介绍的系统模块的参数均是采用模块参数设置对话框进行设置的。用户需要双击打开模块参数设置对话框，然后直接输入数据设置模块参数。在调节参数大小时还需打开模块参数设置对话框设置模块参数，这样做就比较麻烦。解决这个问题的一个有效的方法是使用 MATLAB 工作空间中的变量设置系统模块参数。特别是当系统中有多个模块的参数依赖同一个变量时，使用这种方法非常便利。用户可以直接使用 MATLAB 工作空间中的变量设置模块参数；也可以使用变量表达式设置模块参数。例如，若 k 是定义在 MATLAB 中的变量，则表达式 k, k^2+k, abs(k)等均可以作为系统模块的参数。图 11.1 所示的仿真算例说明

了如何利用 MATLAB 工作空间中的变量设置系统模块参数。仿真系统图中两个增益模块的增益值分别设置为 k 和$\sqrt{k}$，在进行仿真时，这两个模块先从 MATLAB 工作空间读取 k，然后确定出自己的增益。

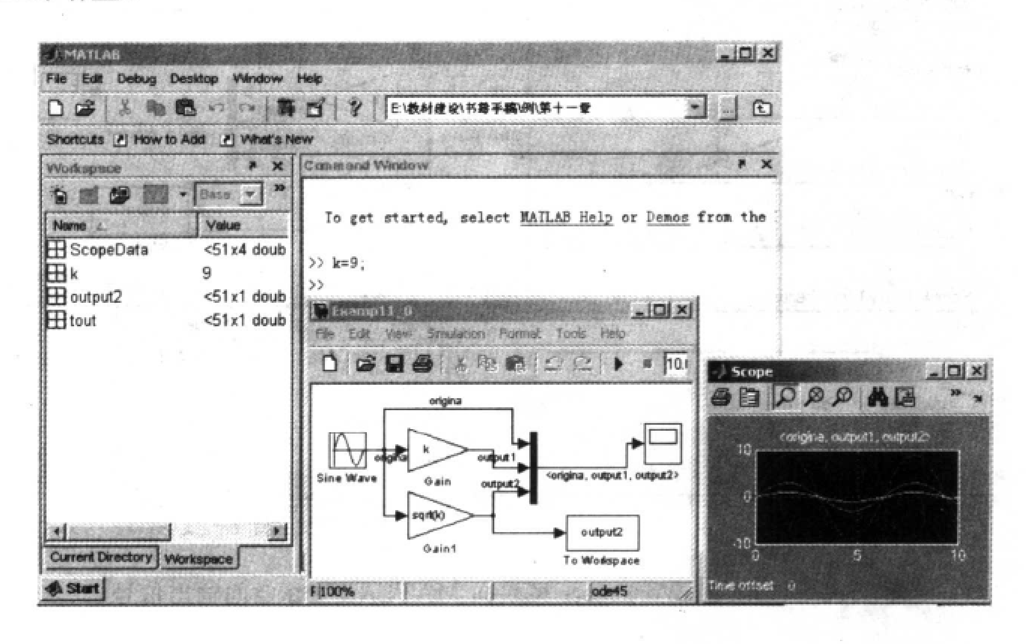


图 11.1　使用 MATLAB 工作空间变量设置模块参数

这里需要说明的一点是，如果系统模块参数设置中使用的变量在 MATLAB 工作空间中没有定义，仿真开始时 Simulink 会提示参数未定义信息。

11.1.2　将信号输出到 MATLAB 工作空间中

前面章节在给出系统仿真结果时，都是使用示波器模块 Scope 输出需要观察的信号的。使用示波器模块可以使用户对输出信号进行简单的定性分析。但当需要对信号做进一步的定量分析或需要将多个信号绘制在一张图中，并分别对各信号进行标注时，用户就需要将系统模型中的相应信号输出到 MATLAB 工作空间中，然后再编写 M 文件进行定量分析。Simulink 提供给用户多种将信号输出到 MATLAB 工作空间的方法，本小节将一一介绍。

1. 使用 Sinks 模型库中的 To Workspace 模块将信号输出至 MATLAB 工作空间

这是一种最直接最方便的将信号输出到 MATLAB 工作空间的方法。图 11.2 给出了这种方法的使用说明。

将需要输出的信号连至 To Workspace 模块的输入端。双击 To Workspace 模块打开该模块的参数对话框，如图 11.2 所示。此对话框中需要设置输出信号的名称、输出数据的点数、输出的间隔和输出数据的类型等。需要指出的是，数据输出类型有 3 种：数组、结构体及带时间变量的结构体。如果用户仅须进行数值分析，则可以简单地设置数据输出类型为数组（见图 11.2 的示例）。

2. 使用示波器 Scope 模块设置将信号输出至 MATLAB 工作空间

只要用户选中 Scope 模块参数设置对话框中包含的 Data History 选项卡中的选项 Save data to workspace，并进行适当的设置，就可以将 Scope 模块显示的信号输出到 MATLAB 工作空间。图 11.3 给出了如何设置 Scope 模块以便将信号输出至 MATLAB 工作空间的方法。

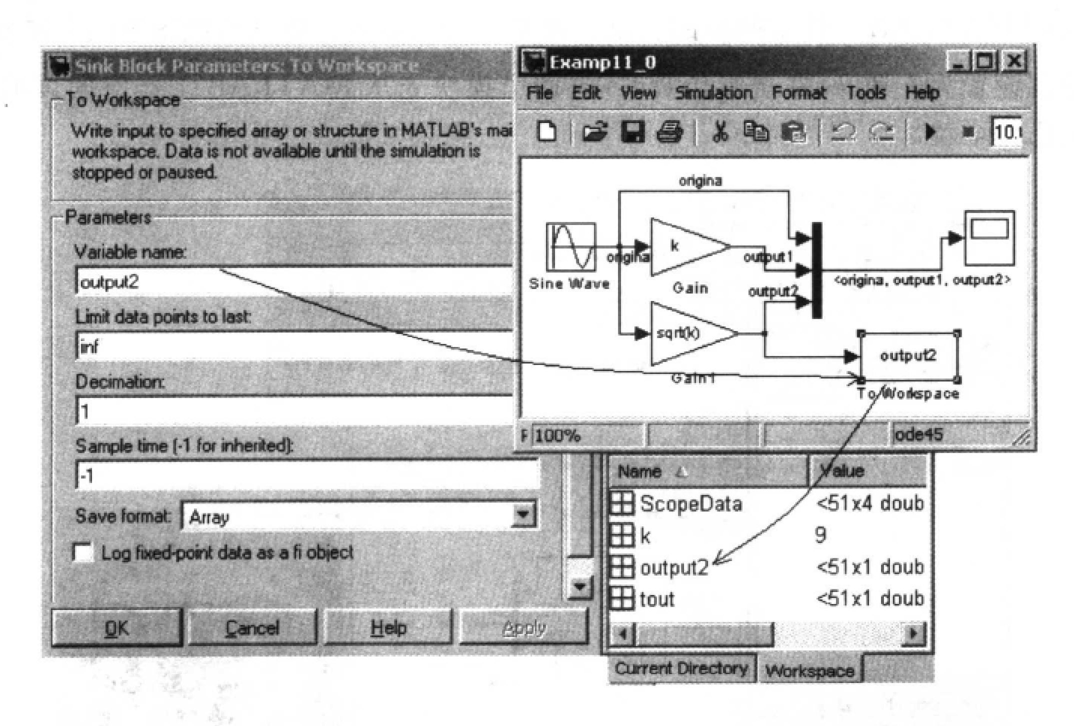


图 11.2　使用 To Workspace 模块向 MATLAB 工作空间输出信号

同使用 To workspace 模块的方法相同，使用 Scope 模块将信号输出至 MATLAB 工作空间的数据保存类型也包含数组、结构体和带时间变量的结构体 3 种。

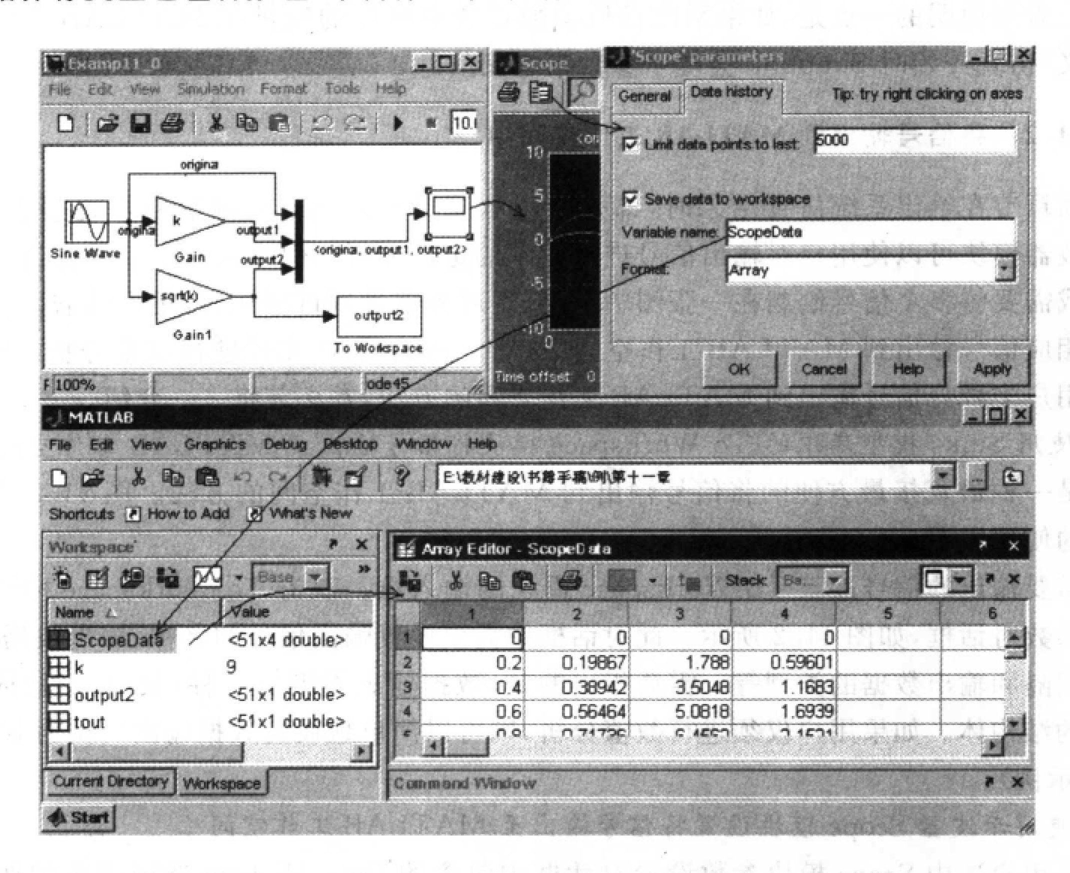


图 11.3　使用 Scope 模块向 MATLAB 工作空间输出信号

3. 使用 Simulation Configuration Parameters 仿真参数设置对话框中的 Data Import/Ex-

port 选项卡设置将信号输出至 MATLAB 工作空间

除了上述的两种将信号从 Simulink 仿真中输出到 MATLAB 工作空间的方法以外，用户还可以使用 Simulation Configuration Parameters 仿真参数设置对话框中的 Data Import/Export 选项卡进行设置。具体的使用方法见例 11.1。

11.1.3 使用 MATLAB 工作空间变量作为系统输入信号

Simulink 与 MATLAB 的信号、数据的交互是相互的，用户除了可以将信号输出到 MATLAB 工作空间以外，还可以使用 MATLAB 工作空间的变量作为系统仿真模型的输入信号。Simulink 系统仿真模型从 MATLAB 工作空间输入信号的方法有两种。

1. 使用 Sources 模型库中的 From Workspace 模块

使用 Sources 模型库中的 From Workspace 模块可以将 MATLAB 工作空间中的变量作为 Simulink 系统仿真模型的输入信号。可以输入到 Simulink 系统仿真模型中的信号的格式如下：

```
>> t=0:time_step:final_time;      %表示信号输入时间范围与时间步长。
>>x=func(t);                      %表示每一时刻的信号值。
>>input=[t',x']                   %表示信号的输入向量。输入变量的第一列必须
                                   是时间序列，后面的各列是信号序列。
```

例如：在 MATLAB 命令窗口中键入下列语句并运行。

```
>> t=0:0.1:10;
>> x=sin(t);
>> input=[t',x'];
```

在系统模型的 From Workspace 模块中使用该变量作为输入信号，如图 11.4 所示。读者可以运行此系统，观察仿真结果，如图 11.5(a)所示。系统输入信号 input 的作用相当于 Sources 模型库中的 Sine Wave 模块。

需要指出的是，From Workspace 模块参数的默认设置是进行插值计算，即 Simulink 会对没有定义的时间点进行线性插值。例如 MATLAB 命令窗口中键入语句：

```
>> t=[0 3 6 9 10];
>> x=[-1 1 -1 1 1/3];
>> input=[t',x'];
```

将生成一个三角波。运行如图 11.4 所示的系统仿真模型，仿真结果如图 11.5(b)所示。可见，Simulink 对 MATLAB 工作空间的输入信号 input 进行了线性插值。

2. 使用 Simulation Configuration Parameters 仿真参数设置对话框中的 Data Import/Export 选项卡设置

使用 Simulation Configuration Parameters 仿真参数设置对话框中的 Data Import/Export 选项卡可以设置使用 MATLAB 工作空间中的变量作为 Simulink 系统仿真模型的输入信号。下面通过举例说明如何使用仿真参数设置 Simulation Configuration Parameters 对话框中的 Data Import/Export 选项卡将数据在 MATLAB 工作空间和 Simulaton 模型中之间进行交互。

例 11.1　建立简单的仿真系统，要求：

(1) 系统输入信号是单位幅值单位频率的正弦信号，由 MATLAB 工作空间中的变量提供；

(2)系统输出是输入的 3 倍，要求输出至 MATLAB 工作空间；

使用 MATLAB 绘图命令绘制系统的输入和输出信号。

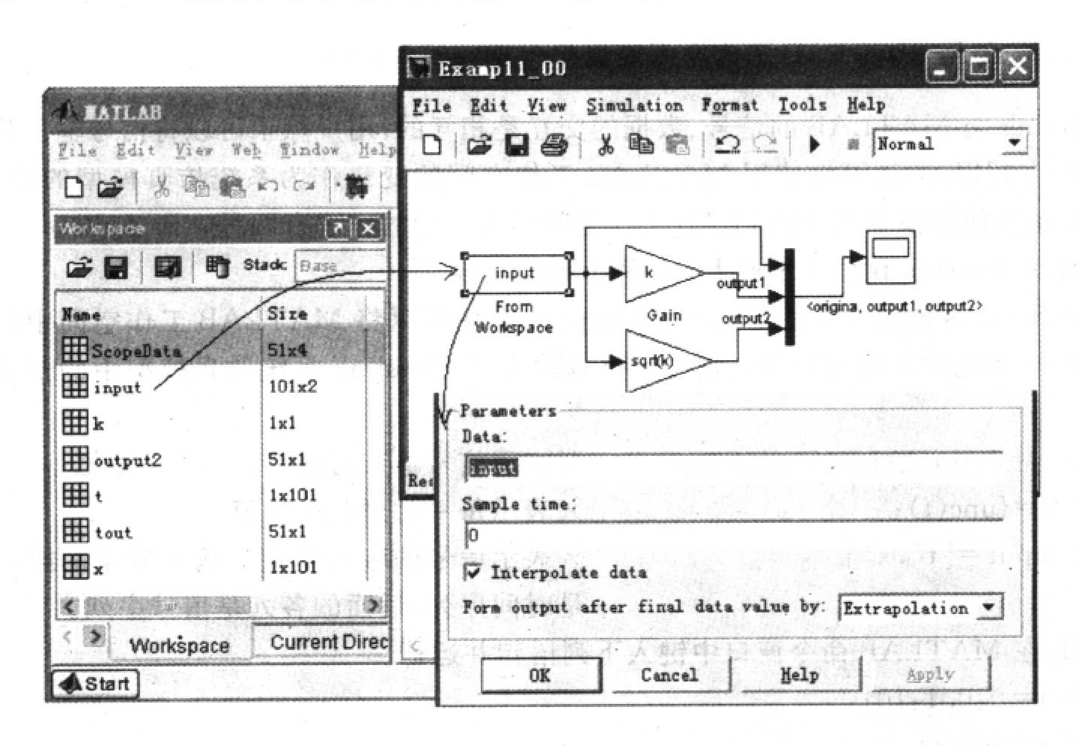


图 11.4　MATLAB工作空间变量作为系统输入信号

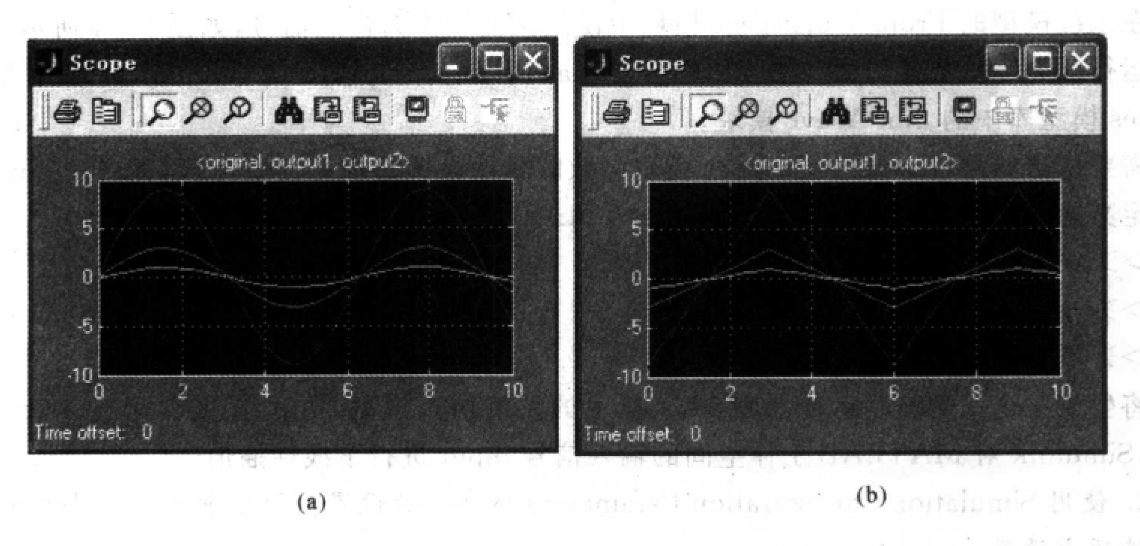


图 11.5　使用 input 信号做输入的仿真结果

(a) 正弦输入的仿真结果；(b) 三角波输入的仿真结果

解　建立系统的 Simulink 仿真模型如图 11.6 所示。Sources 模型库中的 In1 模块和 Sinks 模型库中的 Out1 模块都是虚模块，子系统中使用时，仅仅表示将信号传入或传出子系统。而在最顶层的系统模型中使用时，可以通过它们从 MATLAB 工作空间中输入信号、将计算结果输出到 MATLAB 工作空间中。

为了从 MATLAB 工作空间中输入信号并将计算结果输出到 MATLAB 工作空间中，打开 Simulation Configuration Parameters 仿真参数设置对话框中的 Data Import/Export 选项卡，如图 11.6 所示。

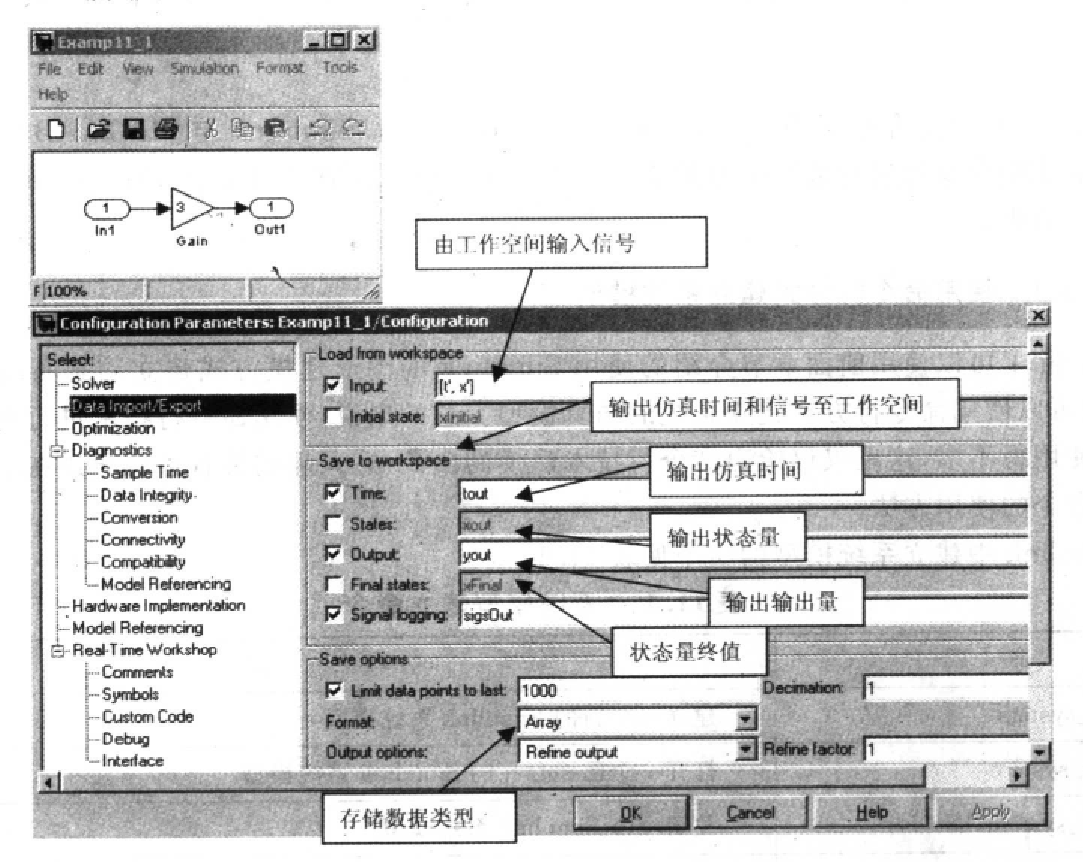


图 11.6　例 11.1 系统仿真模型及其 Data Import/Export 设置

如果 MATLAB 命令窗口中输入并运行命令：

```
>> t=0:0.1:10;
>> x=sin(t);
```

采用默认的系统仿真参数运行系统仿真，使用 MATLAB 绘图命令：

```
>> plot(t,x,tout,yout,'-.')
```

可以绘制出系统的输入信号和输出信号，如图 11.7 所示。

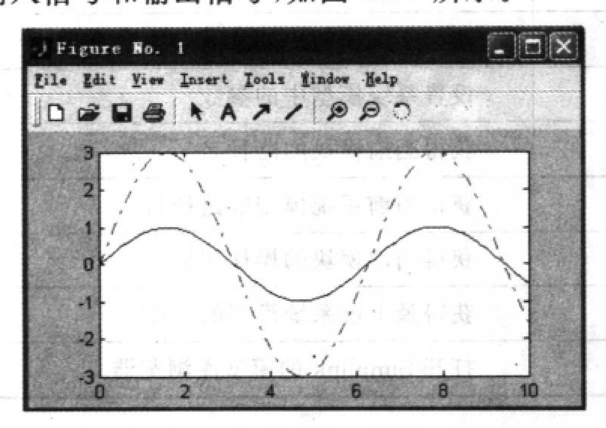


图 11.7　系统输入和输出信号

11.2 使用命令行方式对动态系统进行建模和仿真分析

第11.1节主要介绍了 Simulink 与 MATLAB 的接口，Simulink 与 MATLAB 的数据交互是能够使用命令行进行数字仿真的前提。本节开始介绍如何使用命令行对动态系统进行建模和仿真分析。

11.2.1 使用命令行方式建立系统模型

用户除了可以使用前面章节介绍的使用 Simulink 的图形建模方式建立动态系统的模型之外，也可以使用命令行方式建立系统的仿真模型。总的来说，使用命令行技术进行系统建模的方法使用得不多，这里仅仅给出命令行技术建模的命令，感兴趣的读者可以通过在线帮助了解各个命令的使用方法。

Simulink 中建立系统模型的命令见表 11.1。

表 11.1 系统模型建立命令

命令	功能
new_system	建立一个新的 Simulink 系统模型
open_system	打开一个已经存在的 Simulink 系统模型
close_system, bdclose	关闭一个 Simulink 系统模型
save_system	保存一个 Simulink 系统模型
find_system	查找 Simulink 系统模型、模块、连线及注释
add_block	在系统模型中加入指定模块
delete_block	从系统模型中删去指定模块
replace_block	替代系统模型中的指定模块
add_line	在系统模型中加入指定连线
delete_line	从系统模型中删去指定连线
get_param	获取系统模型中的参数
set_param	设置系统模型中的参数
gcb	获得当前模块的路径名
gcs	获得当前系统模型的路径名
gcbh	获得当前模块的操作句柄
bdroot	获得最上层系统模型的名称
simulink	打开 Simulink 的模型库浏览器

11.2.2　使用命令行方式进行动态系统仿真

使用命令行方式，用户可以编写并运行系统仿真的 M 文件来完成对动态系统的仿真。在 M 文件中，用户可以反复对同一系统在不同的仿真参数或不同的系统模块参数下进行仿真，这样就不需多次打开 Simulink 图形窗口，使用 Start Simulation 命令进行仿真。特别是当需要分析某个参数对系统仿真结果的影响时，用户可以很容易地使用 for 循环自动修改参数值。这样可以方便、快速地分析不同参数值对系统性能的影响。

1. 使用 sim 命令进行动态系统仿真

(1) 调用格式。sim 命令是使用命令行技术进行动态系统仿真分析最常用的命令。其完整的调用格式为

[t,x,y]=sim(model,timespan,options,ut)

[t,x,y1,y2,…,yn]=sim(model,timespan,options,ut)

实际使用时，用户可以省略 sim 命令中的某些设置，MATLAB 对省略的设置采用默认的参数。

sim 命令实现对 model 指定的系统模型按照给定的仿真参数和系统模型参数进行仿真。

(2) 参数说明。仿真过程中所使用的参数包括所有仿真参数对话框设置的参数、MATLAB 工作空间的输入输出选项卡中的设置及采用命令行方式设置的参数和系统模块参数。

sim 命令中，只有参数'model'是必需的，其他仿真参数均被允许设置为空矩阵，此时 sim 命令对不设置的仿真参数使用系统框图决定的默认参数进行仿真计算。sim 命令中设置的参数具有较大的优先级，设置过的参数将取代模型默认的参数。用户须使用 sim 命令中的 options 参数设置所需的参数。下面是各个参数的详细说明：

t：返回仿真时间向量。

x：返回仿真的状态矩阵，排列次序是先连续状态，后离散状态。

y：返回仿真的输出矩阵，其中每一列对应着一个根层次的输出端口（即顶层系统），排列顺序对应端口数字。如果输出端口的结果是向量信号，则它相应的是占有合适的列数。

y_1，y_2，…，y_n：返回模型中 n 个根层次输出端口的输出。

model：需进行仿真的系统仿真模型框图名称。

timespan：系统仿真时间范围（起始时间至终止时间），可以取如下形式：

tFinal：设置仿真终止时间。仿真起始时间默认为 0。

[tStart tFinal]：设置仿真的起始时间（tStart）和终止时间（tFinal）。

[tStart OutputTimes tFinal]：设置仿真的起始时间（tStart）和终止时间（tFinal），并且设置仿真返回的时间向量[tStart OutputTimes tFinal]，其中，tStart，OutputTimes 和 tFinal 必须递增排列。

options：由 simset 命令设置的除了仿真时间外的仿真参数，是一个结构体变量。

ut：表示系统顶层模型的外部可选输入。ut 既可以使用 MATLAB 函数，亦可以使用多个外部输入 ut_1，ut_2，…。其格式必须符合输入信号的要求。具体要求同由 MATLAB 工作空间传递信号至系统模型的格式。

(3) sim 命令的应用。

1)简单仿真命令的应用。下面仍然以例 11.1 说明 sim 命令的应用。

例 11.1 的系统仿真模型要求输入信号是 MATLAB 工作空间的变量,系统的仿真结果也要求输出到 MATLAB 工作空间中。前面在进行系统仿真时,我们首先将输入信号从 MATLAB 命令窗口输入到其工作空间,然后打开系统的 Simulink 仿真模型,使用模型框图中的 start simulation 命令启动的仿真程序,最后再在 MATLAB 命令窗口中键入绘图命令绘制输入输出信号。实际上用户也可以采用命令行仿真技术进行仿真。这样,用户无须打开系统的 Simulink 仿真模型,将输入信号设置、启动仿真程序以及最后绘制输入输出信号三步工作一起完成。用户只需在设置好系统仿真模型后,将下列命令键入 MATLAB 命令窗口并运行即可完成上述 3 个部分的工作:

```
>> t=0:0.1:10;
>> x=sin(t);
>> [tout,xout,yout]=sim('Exampl1_1');    %使用 sim 命令运行系统仿真,采用的仿真参数同例 11.1。
>> plot(t,x,tout,yout,'-.');
```

2)仿真时间设置。在系统仿真过程中,仿真时间设置是非常关键的。对于不同的动态系统,用户关心的系统响应的时间段是不同的。在前面介绍 sim 命令时已经说明了仿真时间 timespan 的 3 种使用形式。根据不同动态系统仿真的不同要求,用户可以选择使用不同的形式进行仿真:

```
[t,x,y]=sim(model, tFinal)
[t,x,y]=sim(model, [tStart   tFinal])
[t,x,y]=sim(model, [tStart OutputTimes tFinal])
```

需要注意的是仿真结束时间 tFinal 必须大于仿真开始的时间 tStart。此外,在默认情况下,由于仿真时刻受到 Simulink 求解器仿真步长的控制,因而系统仿真的输出结果(输出时间、状态及系统输出量)也会受到 Simulink 求解器仿真步长的控制。如果需要在指定的时刻输出系统的仿真结果,则需要使用仿真时间设置的第三种方式,其中[tStart OutputTimes tFinal]表示输出时间向量,是一个递增的行向量。

sim 命令中设置的仿真时间会覆盖 Simulink 仿真参数设置对话框中设置的仿真时间。

这里仍然以例 11.1 说明不同的仿真时间的设置方式及其结果。为了使用户对使用命令行方式设置系统仿真时间范围有一个直观的了解,将使用几种不同的设置对例 11.1 进行仿真,并绘制这几种设置所得的结果以便于读者进一步理解。

MATLAB 命令窗口键入下列命令并运行,用户可以点击工作空间中的变量 tout1,tout2,tout3, tout4 及其他变量来查看 sim 命令的返回结果。

```
>> t=0:0.1:10;
>> x=sin(t);
>> [tout1,xout1,yout1]=sim('Exampl1_1',5);   %仿真时间范围设置为 0 至 5 秒。输出时间向量 tout1 由 Simulink 求解器步长决定,用户可以点击工作空间中的 tout1 变量查看结果。
>> [tout2,xout2,yout2]=sim('Exampl1_1',[1 6]);   %仿真时间范围设置为 1 至 6 秒。输出时间向量 tout2 由 Simulink 求解器步长决定。
>> [tout3,xout3,yout3]=sim('Exampl1_1',[1 2 4 6]);   %仿真时间范围设置为 1
```

至6秒，且只在tout3=[1 2 4 6]秒时有返回输出。

>> [tout4,xout4,yout4]=sim('Examp11_1',[1:0.2:6]);　%仿真时间范围设置为1至6秒，且每隔0.2秒输出一次。输出时间向量tout4=[1 1.2 1.4 ... 5.6 5.8 6]。

```
>> subplot(221),plot(t,x,tout1,yout1,'.')      %同幅图中绘制系统仿真结果。
>> legend('t-x','timespan=5')                  %标注图形。
>> subplot(222), plot(t,x,tout2,yout2,'.')
>> legend('t-x','timespan=[1 6]')              %标注图形。
>> subplot(223), plot(t,x,tout3,yout3,'.')
>> legend('t-x','timespan=[1:6]')              %标注图形。
>> subplot(224), plot(t,x,tout4,yout4,'.')
>> legend('t-x','timespan=[1:0.2:6]')          %标注图形。
```

系统在不同仿真时间设置下的仿真计算结果如图11.8所示。从MATLAB工作空间的变量列表中也可以查看不同仿真时间设置下，系统仿真各个返回值。例如，由于命令[tout1, xout1,yout1]=sim('Examp11_1',5)没有规定返回时间向量的值，因而MATLAB工作空间的变量列表中tout1的间隔由仿真步长决定；而命令[tout3,xout3,yout3]=sim('Examp11_1',[1 2 4 6])规定了返回的时间向量仅为[1 2 4 6]，因而返回的时间向量tout3和输出向量yout3均是4×1的向量。

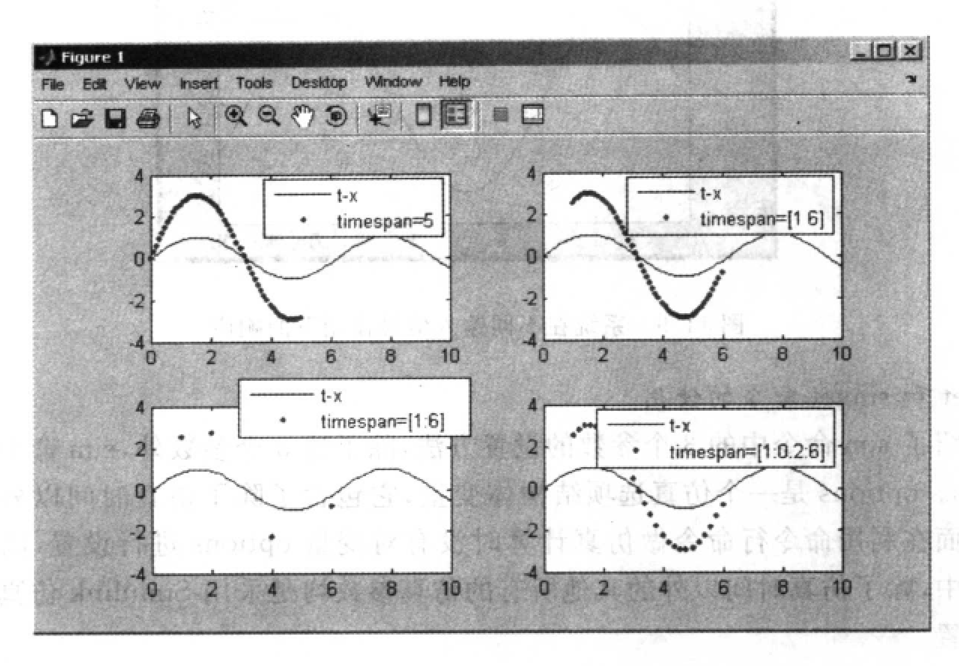


图11.8　不同仿真时间设置下，例11.1系统的仿真结果比较

3)外部输入变量设置。通过适当的设置命令行命令[t,x,y]=sim(model,timespan,options, ut)中的ut，用户也可以使用Sim命令从MATLAB工作空间中获得系统的输入信号。

ut的设置同前面介绍的设置Simulink参数设置对话框的Data Import/Export选项卡的情况相同。ut必须是具有两列的矩阵，其中第一列表示外部输入信号的时刻，第二列表示与给定时刻对应的输入信号的取值。使用矩阵ut可以为系统模型最顶层的In1模块提供外部输入信号，并将自动覆盖Simulink仿真参数设置对话框中Data Import/Export选项卡的

设置。

这里仍以例 11.1 为例说明使用 sim 命令，设置 sim 命令参数使得模型可以从 MATLAB 工作空间获得输入信号的方法。在 MATLAB 命令窗口中键入并运行下列命令：

```
>> t=0:0.1:10;
>> x=sin(t);
>> ut=[t' x'];
>> [tout1,xout1,yout1]=sim('Examp11_1',10,[],ut);          %正弦信号输入下。
>> x1=cos(t);
>> [tout2,xout2,yout2]=sim('Examp11_1',10,[],[t' x1']);    %余弦信号输入下。
>> plot(tout1,yout1,tout2,yout2,'-.')
>> legend('sin(t)','cos(t)')
```

系统在正弦信号和余弦信号作用下的仿真结果如图 11.9 所示。由本例可以看出，设置 sim 命令的 ut 参数，用户可以方便地使用不同的输入信号。

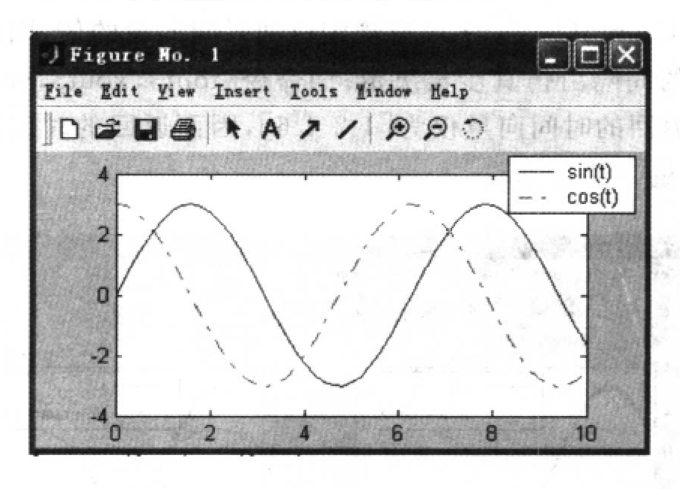


图 11.9　系统在不同输入信号作用下的响应

2. simset 和 singet 命令的使用

前面介绍了 sim 命令中的 3 个参数的设置方法，除了这 3 个参数外，sim 命令中还有一个参数 options。options 是一个仿真选项结构体变量，它包含了除了仿真时间以外的所有的仿真参数。前面在利用命令行命令做仿真计算时没有对变量 options 进行设置，因而实际上在仿真的过程中，除了仿真时间以外的其他所有的仿真参数均是采用 Simulink 仿真参数设置对话框中的设置。

用户可以使用 simset 命令设置结构体变量 options。但为了使用户对此结构体变量有一个总体的了解，首先使用 simget 命令获得表示系统仿真参数的结构体变量。在 MATLAB 命令窗口键入：

```
>> options=simget('Examp11_1')   % 获得系统模型 Examp11_1 的仿真参数选项。
options =
            AbsTol: 'auto'
                    Debug: 'off'
```

```
            Decimation: 1
          DstWorkspace: 'current'
        FinalStateName: ''
             FixedStep: 'auto'
          InitialState: []
           InitialStep: 'auto'
              MaxOrder: 5
    SaveFormat: 'Array'
         MaxDataPoints: 1000
               MaxStep: 'auto'
               MinStep: 'auto'
          OutputPoints: 'all'
       OutputVariables: 'ty'
                Refine: 1
                RelTol: 1.0000e-003
                Solver: 'ode45'
          SrcWorkspace: 'base'
                 Trace: ''
             ZeroCross: 'on'
    ExtrapolationOrder: 4
NumberNewtonIterations: 1
```

由此可以看出使用 simget 命令获得的结构体变量包含了除仿真时间以外的所有仿真参数选项。这些仿真参数选项均可以使用 simset 命令进行设置。下面首先对常用的仿真参数选项及其取值做简单的介绍。

(1) 仿真参数选项介绍。

AbsTol:表示绝对误差限,取值为标量,缺省值为 1e-6。仅用于变步长求解器。

Decimation:表示系统仿真结果返回数据点的间隔,取值为正整数,缺省值为 1。值为 1 表示每一个仿真结果数据均返回到相应的变量中;值为 2 表示仿真结果每隔一个数据点返回到相应的变量中,依此类推。

FixedStep:表示定步长求解器的步长,取值为正数,标量。如果对离散系统进行仿真,其缺省值是离散系统的采样周期;如果对连续系统求解,其缺省值是仿真时间范围的 1/50。

InitialState:表示系统的初始状态,取值向量,缺省值是空向量。若系统中同时存在连续状态和离散状态,则此向量的次序是先连续状态的初值,后离散状态的初值。初始状态的设置会覆盖系统模型中默认的状态初始值。

InitialStep:表示系统仿真的初始步长(估计值),仅用于变步长求解器。在仿真时首先采用估计的步长,缺省时由求解器决定初始仿真步长。

MaxStep:最大步长,取值为正数标量,缺省值时 auto。仅用于变步长求解器,缺省时最大仿真步长是仿真时间范围的 1/50。

RelTol:表示相对误差限,取值是正数标量,缺省值是 1e-3,仅用于变步长求解器。

Solver：表示 Simulink 的求解器，其取值是由"|"隔开的字符串：VaribleStepDiscrete | ode45 | ode23 | ode113 | ode15s | ode23s | FixedStepDiscrete | ode5 | ode4 | ode3 | ode2 | ode1。缺省值是变步长连续求解器，计算方法是 ode45(Dormand - Prince)。

ZeroCross：表示仿真过程中的过零检测，取值为 on 或 off，缺省值为 on，仅用于变步长求解器。on 表示对系统模块进行过零检测，而 off 表示不进行过零检测。

(2) simset 命令的用法。simset 命令是专门用来设置结构体变量 options 的，即是用来设置系统仿真参数的。其调用格式有 3 种，分别为

```
options=simset('name1',value1,'name2',value2,…);
options=simset(oldopts, 'name1',value1,…);
options=simset(oldopts, newopts);
```

使用说明：

options=simset('name1',value1,'name2',value2,…)：设置指定的仿真参数选项值。其中 name 为指定的仿真参数，value 为指定的取值。

options=simset(oldopts, 'name1',value1,…)：修改仿真参数结构体变量中已经存在的指定仿真参数选项。其中，oldopts 表示已经存在的结构体。

options=simset(oldopts, newopts)：合并两个已经存在的结构体变量，并使用 new_opstruct 中的域值覆盖 old_opstruct 中具有相同域名的域值。

例如：若用户希望使用命令行命令关闭例 11.1 描述的系统的仿真过零检测之后再进行仿真计算，只需在 MATLAB 命令窗口键入下列命令即可：

```
>> ex11_1_options=simset('ZeroCross','off');    %关闭系统的仿真过零检测。
>> [tout,xout,yout]=sim('Exampl1_1',10,ex11_1_options);    %使用 ex11_1_options 仿真参数选项进行系统仿真。
```

(3) simget 命令的用法。simget 命令是用来获得指定系统模型的仿真参数设置的命令。其调用格式为

```
struct=simget(model);
value=simget(model,property)
value=simget(OptionStructure,property)
```

其中：

struct=simget(model)：可以获得指定系统模型的所有仿真参数设置结构体变量；

value=simget(model, property)：可以获得指定系统模型的指定仿真参数 property 的取值；

value=simget(OptionStructure,property)：可以获得系统仿真参数选项中指定的仿真参数的取值。变量 property 可以是一个包含多个系统仿真参数元胞数组，此时返回值也是元胞数组。

3. simplot 命令的使用

前面在进行动态系统仿真时，很多时候是利用 Sinks 模型库中的 Scope 模块观察系统的仿真结果的。通过对 Scope 模块的一些操作，用户可以方便地观察系统的输出信号。而利用 plot 命令绘制的图形没有利用 Scope 模块表现的信号那样直观和容易操作，因此 MATLAB 给用户提供了另一个绘图命令——simplot。simplot 命令绘制的图形和 Scope 模块输出的图

形类似。

simplot 命令的调用格式是：

simplot(data)；

simplot(time,data)

其中，time 表示动态系统仿真结果的输出时间向量。当系统输出数据为带有时间向量的结构体变量时，此参数将被忽略；

data 是动态系统仿真结果的输出数据(MATLAB 工作空间中已有的数据变量)。若希望将几个信号绘制在一张图中以便于对它们进行比较，必须先使用命令 data = {signal1 signal2 ……}组合数据。

仍然以例 11.1 为例，在 MATLAB 键入如下命令可以将 3 种不同信号作用下系统的输出信号绘制在一张类似 Scope 模块显示的图形中，如图 11.10 所示。

```
>> t=0:0.1:10;
>> x=sin(t);
>> ut=[t' x'];
>> [tout1,xout1,yout1]=sim('Exampl1_1',10,[],ut);    %正弦信号输入下。
>> x1=cos(t);
>> [tout2,xout2,yout2]=sim('Exampl1_1',10,[],[t' x1']);   %余弦信号输入下。
>> x3=sin(t).*cos(t);
>> [tout3,xout3,yout3]=sim('Exampl1_1',10,[],[t' x3']);   %输入的是单位正弦
                                                             信号和余弦信号的
                                                             乘积。
>> data = {yout1,yout2,yout3};                               %组合信号。
>> simplot(tout1,data);    %使用 simplot 命令将 3 个信号绘制在一张图中。
```

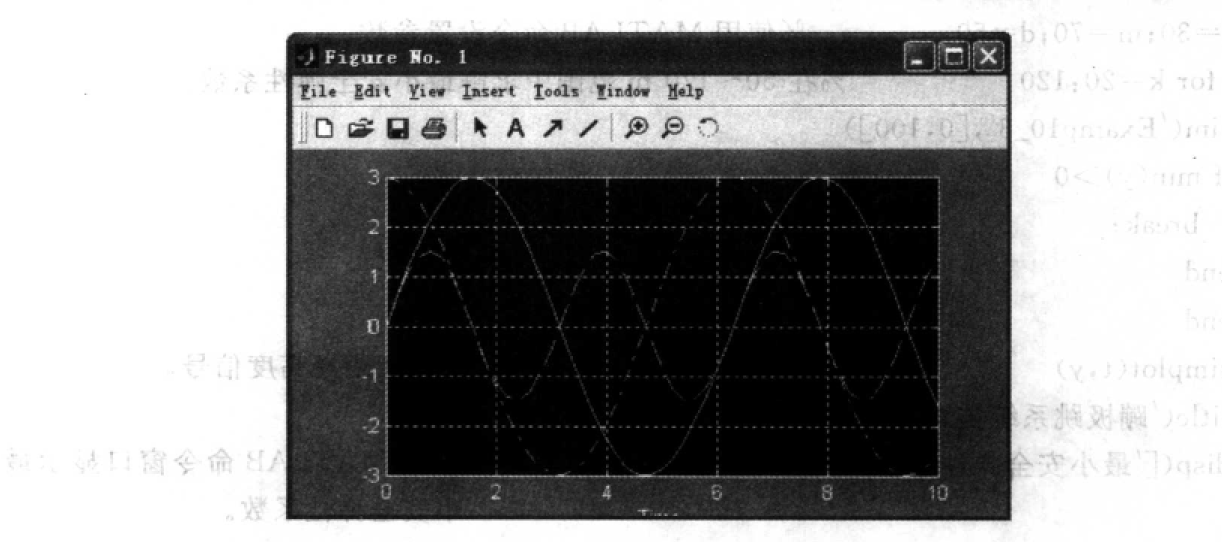


图 11.10　使用 simplot 命令绘制的系统仿真结构

11.3　使用 MATLAB 脚本文件分析系统

既然至此用户已经能够使用命令行技术对动态系统进行仿真研究和分析了，而且 MATLAB 的脚本文件是一系列的命令的集合，是由一系列 MATLAB 命令、内置函数及 M 文件等组成的文件，因此，用户可以考虑将这些命令行命令集中起来以形成脚本文件，然后使用已形成的脚本文件对动态系统进行仿真研究。

本节将通过实例来说明如何使用 MATLAB 脚本文件实现对动态系统的仿真分析。

例 11.2　在第十章例 10.3 中曾经用 S-函数对蹦极跳系统做了仿真分析。蹦极跳系统的仿真模型及仿真结果如图 10.8 所示。由仿真结果可知，例 10.3 所描述的蹦极跳系统对于质量为 70 kg 重的蹦极者来说是非常危险的，因为仿真结果显示蹦极者有触地的危险，他与地面的距离小于 0。为了满足大体重的蹦极爱好者的要求，必须对系统参数做适当的调整。

很显然，加大弹性绳索的弹性系数可以减少蹦极者触地的危险性，当弹性系数加大到一定的程度，蹦极跳系统对 70 kg 质量的蹦极者就是安全的了。下面编写 MATLAB 脚本文件对不同弹性系数情况下蹦极跳系统进行仿真分析，求出对质量为 70 kg 蹦极者来说最小的安全的弹性系数。

解　系统仿真模型仍然使用例 10.3 构建的模型，该模型使用 S-函数实现蹦极跳系统，需要 4 个参数：弹性绳索长度 l 和弹性系数 k、蹦极者质量 m、桥梁距地高度 d。在求最小安全弹性系数时，需保持除了弹性系数 k 以外的其他 3 个参数不变，即各个参数的取值仍然为 $m=70$ kg，$l=30$ m，$d=50$ m。

编写 MATLAB 脚本文件 Exampl1_2_cmd.m 以求取最小的安全弹性系数，程序如下：

```
l=30;m=70;d=50;              %使用 MATLAB 命令设置参数。
 for k=20:120               %在 50～120 m 范围中求解最小安全弹性系数。
sim('Examp10_3',[0,100])
if min(y)>0
  break;
end
end
simplot(t,y)                %绘制最小安全弹性系数下蹦极者距地高度信号。
title('蹦极跳系统仿真')
disp(['最小安全弹性系数',num2str(k),'kg/m'])      %在 MATLAB 命令窗口显示最
                                                  小安全弹性系数。
dis=min(y);                 %求取最小安全弹性系数下蹦极者距地的最小高度。
disp(['蹦极者距地最短距离为',num2str(dis),'m'])
```

在 M 文件编辑器中键入并运行上述程序后，MATLLAB 命令窗口会显示对于质量为 70 kg 的蹦极者来说最小的安全弹性系数和最小安全弹性系数下蹦极者距地的最小高度：

　　最小安全弹性系数 109kg/m；

蹦极者距地最短距离为 0.10554m；

同时，simplot 命令也为用户绘制了最小安全弹性系数 109kg/m 下，蹦极者距地距离的曲线，如图 11.11 所示。

通过仿真算例 11.2 说明了如何使用 MATLAB 脚本文件进行动态系统仿真。笔者认为使用脚本文件进行仿真非常方便。表现在以下几个方面：

(1) 能够自动重复地运行仿真；

(2) 在仿真过程中可以动态地调整参数，本例使用循环语句改变参数的值；

(3) 可以方便地分析不同输入信号作用下系统的响应。

因此，由于这几方面的便利，笔者近几年来对实际的工程系统进行调参和仿真分析时均是使用脚本文件对系统进行分析的。

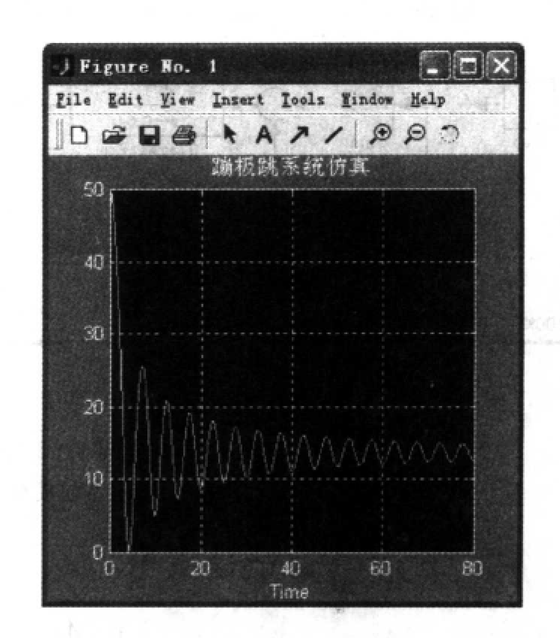


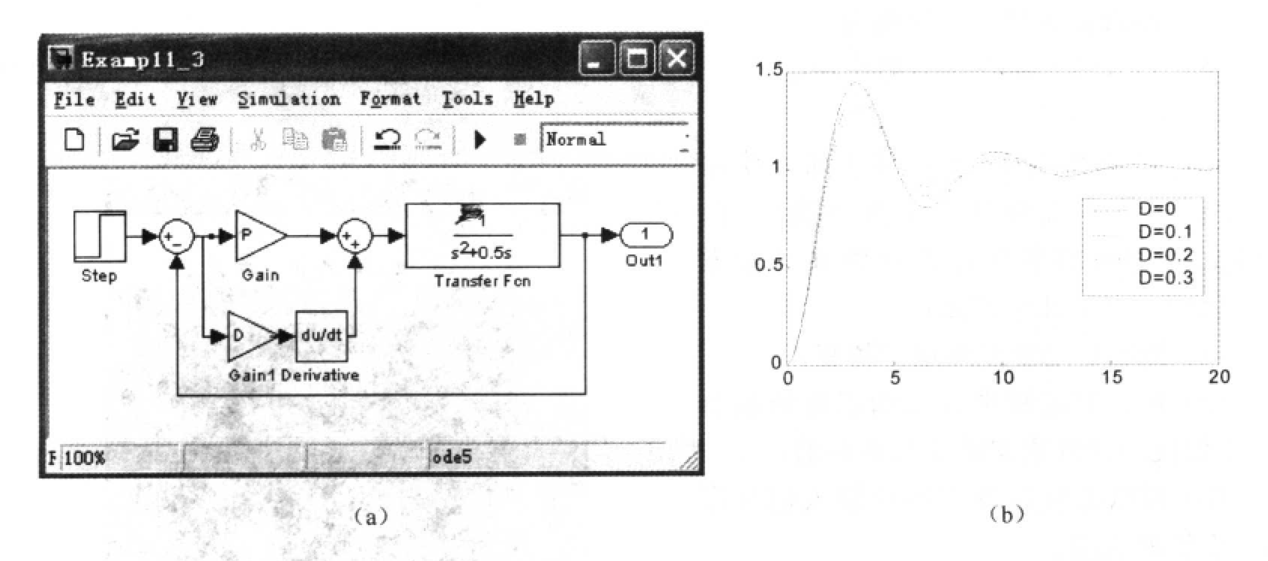
图 11.11 最小安全弹性系统下系统的仿真结果

例 11.3 图 11.12(a)所示是使用比例加微分控制的控制系统，其中 P 和 D 分别是控制器的比例系数和微分系数。分析当 $P=1$ 时，系统在单位阶跃信号作用下，微分系数 D 分别取 0，0.1，0.2 及 0.3 时控制系统的响应，将几种情况所得的仿真结果绘制在一张图中进行比较，分析微分控制的特点。

解 编写的 M 文件程序为

```
P=1;
yy=[];
for D=0:0.1:0.3                          %微分系数取不同的值。
    sim('Examp11_3',20)
    yy=[yy yout];                        %组合微分系数取不同值时得到的系统输出值。
end
plot(tout,yy(:,1),'k-', tout,yy(:,2),'k--', tout,yy(:,3),'k-.', tout,yy(:,
4),'k:');
legend('D=0','D=0.1','D=0.2','D=0.3')
```

从上两例可以看出，使用 MATLAB 脚本文件可以非常方便地分析系统中的某些参数发生变化对系统性能的影响。从仿真结果中可以看出，对于比例+微分控制来说，当比例系数保持不变时，微分系数取值越大，系统的阻尼越大，这样系统的超调量就越小。



(a) (b)

图 11.12 例 11.3 的系统模型及其仿真结果

(a) 系统模型； (b) 仿真结果

11.4 回调函数

11.4.1 回调函数基本概念

所谓回调函数，是指系统模型或系统模型中的某些模块在特定的时刻、发生特定的行为时所运行的一系列用户自定义的命令集合。为了使用户更容易接受回调函数的概念及其能够完成的功能，先来看看 MATLAB 提供的使用回调(callback)函数的典型示例。

请读者在 MATLAB 命令窗口键入：

```
>> f14;
```

即可打开 MATLAB/Simulink 提供的复杂的仿真模型 f14。读者无须了解该模型的作用及各个模块组成的原理。点击该模型中的某些模块，读者会发现 f14 中许多模块的参数对话框中使用的是变量，而在模型中却找不到任何关于这些变量的定义。但当读者点击 Start simulation 图标▶时，f14 模型却能够顺利地运行并给出正确的仿真结果。

难道真的不需要在运行仿真之前事先定义变量吗？答案是否定的。相信每个用户在建模做仿真时都曾经有过忘记定义变量的经历。如果用户建立了一个系统仿真模型，只要该系统中仅有一个变量没有定义，在启动仿真运行时，Simulink 都会给出错误提示。事实上，f14 仿真模型中用到的一些变量已经事先定义好了，只是现在定义变量的方式和以前我们使用的方式不一样而已。对于这一点，用户可以从 MATLAB 工作空间看出，实际上在打开 f14 模型之后，这些参数已经存在于 MATLAB 工作空间中了。在 MATLAB 命令窗口中键入：

```
>> who
```

用户就可以查询到所有这些变量了。

那么，这些变量是怎样被赋值的？下面就来回答这个问题。事实上，f14 系统仿真模型被

打开时，MATLAB 就自动调用执行了某些 MATLAB 命令和函数，完成了对所需参数的定义，这些 MATLAB 命令及 MATLAB 函数的集合就是 f14 的回调函数。

从某种意义上来说，回调函数就像许多高级编程语言中的事件处理程序。在 MATLAB 中，为模型或模块的某种行为设置回调函数的方法是将该行为对应的参数（称为模型的回调参数）的值设置为需要执行的回调函数名。回调参数就像高级程序语言中的事件。那么剩下的问题就是 MATLAB 中哪些行为对应哪些回调参数？这些回调参数值对应的回调函数会在什么时间调用？表 11.2 列出了部分回调参数及对应的回调函数的执行时间。

表 11.2　模型的回调函数及其对应的回调函数执行时间

回调参数名称		回调函数执行时间
模型回调参数	CloseFcn	系统模型框图关闭之前执行
	PostLoadFcn	系统模型加载完成后执行。当编写一个要求模型完全加载后方能启动的界面程序时非常有用
	InitFcn	系统模型仿真开始时执行
	PostSaveFcn	系统模型保存后执行
	PreLoadFcn	系统模型加载前执行
	PreSaveFcn	系统模型保存前执行
	StartFcn	系统仿真开始前执行
	StopFcn	系统仿真结束后执行。StopFcn 执行前，系统的仿真结果先被输出到 MATLAB 工作空间或数据文件中了
系统模块回调参数	CloseFcn	当使用 close_system 命令关闭时执行
	CopyFcn	系统模块被复制后执行。这个回调对子系统是递归的
	DeleteFcn	模块被删除前执行。这个回调对子系统是递归的
	DestroyFcn	系统模块被清除后执行
	InitFcn	系统框图被编译以及模块参数被估值前执行
	LoadFcn	系统框图加载后执行。这个回调对子系统是递归的
	ModelCloseFcn	系统框图关闭前执行。这个回调对子系统是递归的
	MoveFcn	系统模块移动或调整大小时执行
	NameChangeFcn	模块的名称或路径改变后执行。这个回调对子系统是递归的
	OpenFcn	模块打开时执行。此参数一般用于子系统模块。在用户双击打开模块或使用以该模块为参数的 open_system 命令时执行。
	ParentCloseFcn	关闭包含此系统模块的子系统之前或作为使用 new_system 命令建立的新子系统中的一部分时执行
	PreSaveFcn	系统框图保存前执行。这个回调对子系统是递归的
	PostSaveFcn	系统框图保存后执行。这个回调对子系统是递归的
	StartFcn	系统框图被编译之后，系统仿真开始之前执行
	StopFcn	在系统仿真以任何形式终止的时候执行

11.4.2　回调函数的使用

对于回调函数,用户应该掌握两个比较重要的命令:

```
set_param('模型名称','回调参数','回调函数名')
get_param('模型名称','回调参数')
```

1. get_param 命令

get_param 命令用于获得系统仿真模型的某个回调参数对应的回调函数名。其调用格式是

```
get_param('模型名称','回调参数')
```

例如在 MATLAB 命令窗口键入

```
>> f14                    %必须先打开 f14 模型。
>> get_param('f14','PreLoadFcn')        %使用 get_param 命令。
ans =
f14dat
```

由此可见,使用 get_param 命令得到了 f14 模型的回调参数 PreLoadFcn 对应的回调函数是名为 f14dat 的 M 文件。那么,在 f14 模型加载前,MATLAB 先执行了 f14dat.m 文件,实现了对仿真所需参数的设置。感兴趣的读者可以在 MATLAB 命令窗口键入:

```
>> open f14dat
```

打开 f14dat.m 文件,可以看到此文件是给仿真所需的各个参数赋值的。

2. set_param 命令

set_param 命令用于设置系统仿真模型的某个回调参数的值(即需执行的回调函数的名称)。其调用格式是

```
set_param('模型名称','回调参数','回调函数名')
```

这里使用一个较简单的例子来说明回调函数的使用。系统的仿真模型如图 11.13 所示。系统中有两个参数需要在仿真前被赋值,否则无法进行仿真计算。现在编写两个参数的脚本文件。

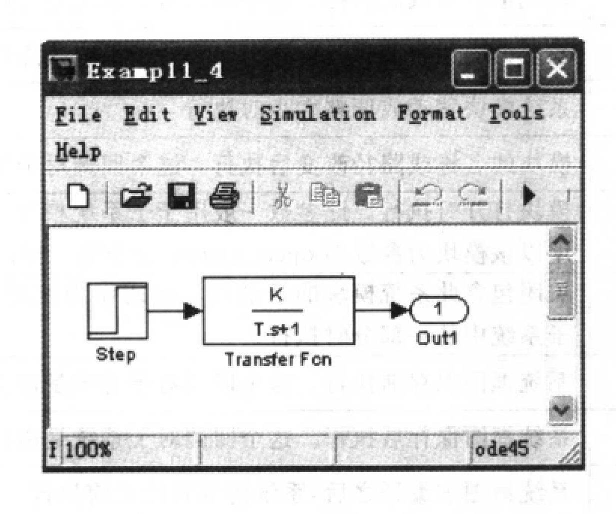


图 11.13　仿真模型

第一个脚本文件完成对这两个参数的幅值，并保存为 kvalue. m，其程序为

K=2；T=2.5；

第二个脚本文件完成对仿真结果的处理工作，即将仿真结果绘图，并保存为 out_graphic. m，程序中的语句是

simplot(tout，yout)；

然后请读者在 MATLAB 命令窗口键入命令：

```
>> set_param('Examp11_4','PreLoadFcn','kvalue')      %加载模型时，执行赋值程序 kvalue. m。
>> set_param('Examp11_4','StopFcn','out_graphic')    %仿真结束时，执行绘图程序 out_graphic. m。
```

打开系统模型 Examp11_3，此时读者可以发现 MATLAB 工作空间已经保存了运行仿真所需的两个参数 K 和 T，说明赋值程序 kvalue. m 已经被执行，启动 Start simulation 命令，仿真结束后，MATLAB/Simulink 自动调用了绘图程序 out_graphic. m 绘制了模型输出的波形。

习　题

11.1　某单位反馈二阶系统的开环传递函数是 $G(s)=\dfrac{1}{s(s+0.5)}$，现预加比例微分控制改善其性能。建立系统的仿真模型如图 11.14 所示，试使用命令行方式对系统进行仿真分析。要求：

(1) 系统输入为单位阶跃信号，阶跃时刻为 0。

(2) 分析系统在比例系数 $K_p=1$，微分系数 K_d 分别取 0.1，0.2，0.3 及 0.4 时控制系统的响应，并绘制响应曲线进行比较，分析微分控制的特点。

(3) 分析系统在微分系数 $K_d=0.2$，比例系数 K_p 分别取 0.5，1.0，1.5 及 2.0 时控制系统的响应，并绘制响应曲线进行比较，分析比例控制的特点。

(4) 当 $K_p=0:0.5:2$，$K_d=0:0.1:0.5$ 时，求系统超调量最小时对应的 K_p 和 K_d。

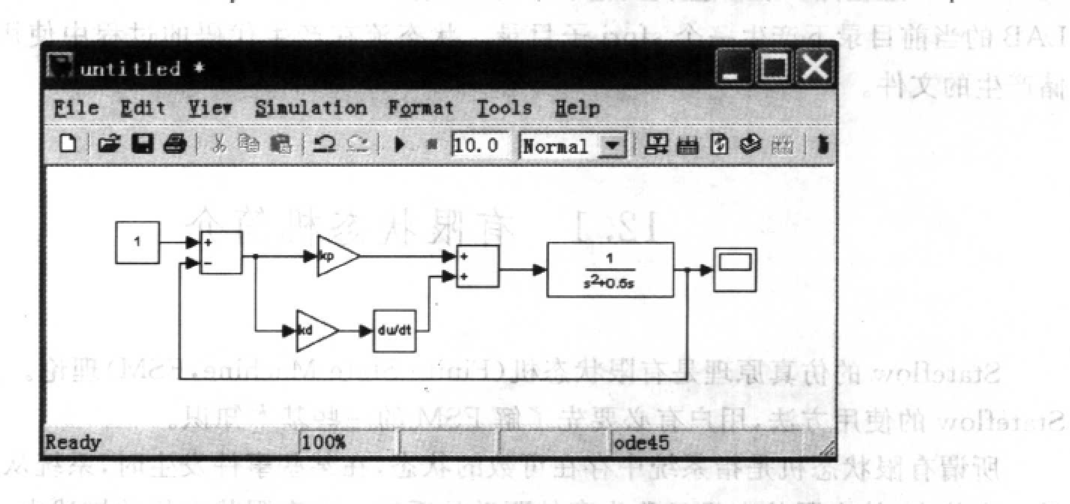


图 11.14　题 11.1 图

第十二章　利用状态流 Stateflow 进行控制系统状态转换

Statefolw 是一种图形化的设计开发工具，是有限状态机的图形实现工具，有人称之为状态流。主要用于 simulink 中控制和检测逻辑关系的表示。用户可以在进行 Simulink 仿真时，使用这种图形化的工具实现各个状态之间的转换，解决复杂的监控逻辑问题。它和 Simulink 同时使用使得 Simulink 更具有事件驱动控制能力。利用状态流可以做以下事情：

(1) 对基于有限状态机理论的相对复杂系统进行图形化建模和仿真；

(2) 设计开发确定的、检测的控制系统；

(3) 更容易在设计的不同阶段修改设计、评估结果和验证系统的性能；

(4) 自动直接地从设计中产生整数、浮点和定点代码(需要状态流编码器)；

(5) 更好地结合利用 MATLAB 和 Simulink 的环境对系统进行建模、仿真和分析。

在状态流图中，利用状态机原理、流图概念和状态转化图，状态流能够对复杂系统的行为进行清晰、简洁的描述。

Stateflow 生成的监控逻辑可以直接嵌入到 Simulink 模型下，两者之间能够实现无缝链接。仿真初始化时，Simulink 会自动启动编译程序，将 Stateflow 绘制的逻辑框图转换成 C 格式的 S-函数(Mex一文件)，产生的代码就是仿真目标，且在状态流内称作 Sfun 目标，这样在仿真过程中直接调用相应的动态链接库文件，将二者组成一个仿真整体。Sfun 目标只能与 Simulink 一起使用。在产生代码前，如果还没有建立名为 sfprj 的子目录，状态流会在 MATLAB 的当前目录下产生一个 sfprj 子目录。状态流在产生代码的过程中使用 sfprj 子目录存储产生的文件。

12.1　有限状态机简介

Stateflow 的仿真原理是有限状态机(Finite State Machine，FSM)理论。为了更快地掌握 Stateflow 的使用方法，用户有必要先了解 FSM 的一些基本知识。

所谓有限状态机是指系统中存在可数的状态，在某些事件发生时，系统从一个状态转换成另一个状态，故有限状态机又称为事件驱动的系统。在有限状态机的描述中，可以设计出由一种状态转换至另一种状态的条件，并可对每对可转换的状态均设计状态迁移事件，从而构造出状态迁移图。

Simulink/Stateflow 为用户提供了图形界面支持的设计有限状态机的方法。它允许用户

建立有限的状态，用图形的形式绘制出状态迁移的条件，并使用其规定的命令设计状态迁移执行的任务，从而构造出整个有限状态机系统。

在 Stateflow 中，状态和状态转换是最基本的元素，有限状态机的示意图如图 12.1 所示。

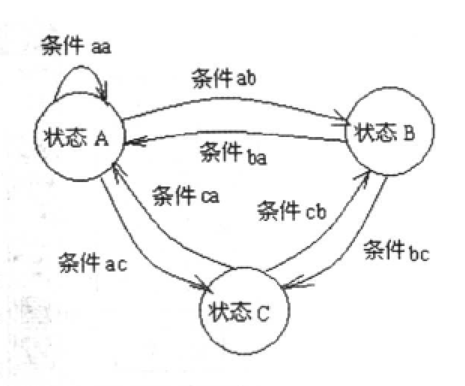


图 12.1　有限状态机示意图

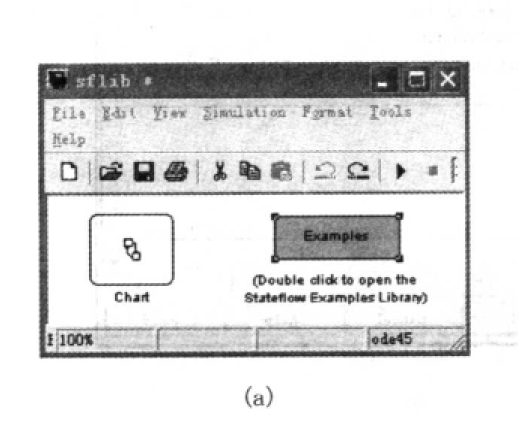
图 12.1 所示有 3 个(有限)状态，这几个状态的转换是有条件的，其中有些状态之间是相互转换的，A 状态是自行转换的。在有限状态机系统中，还表明了状态迁移的条件或事件。

Stateflow 模型一般是嵌在 Simulink 模型下运行的，Stateflow 是事件驱动的，这些事件可以来自同一个 Stateflow 图中，也可以来自 Simulink。

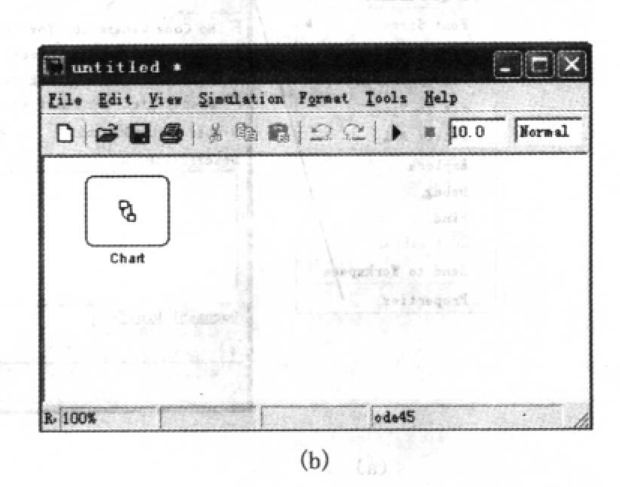
12.2　Stateflow 应用基础

在 MATLAB 命令窗口键入 Stateflow 命令，将打开如图 12.2(a)所示的界面。其中，sflib 窗口中有许多 Simulink 为用户提供的仿真算例。

如果用户在 MATLAB 命令窗口键入 sfnew 命令，将打开嵌入 Stateflow 模块 Charts 的 Simulink 窗口 untitled＊，如图 12.2(b)所示。其中，Chart 是空白的 Stateflow 模块图标。



(a)



(b)

图 12.2　Stateflow 启动窗口

(a) sflib 窗口；(b) 嵌入 Stateflow 模块的 Simulink 窗口

双击 untitled＊窗口中的 Stateflow 模块，打开如图 12.3 所示的 Stateflow 编辑界面，用户可以在此窗口中编辑所需的 Stateflow 模型。Stateflow 提供了强大的图形编辑功能，用户可以使用它描述很复杂的逻辑关系式。

Stateflow 编辑界面中点击鼠标右键，可以看到如图 12.4(a)所示的快捷菜单，选择其中的 Properties(属性)菜单，可以打开如图 12.4(b)所示的对话框，用户可以在此对话框中设置整个 Stateflow 模型的属性。

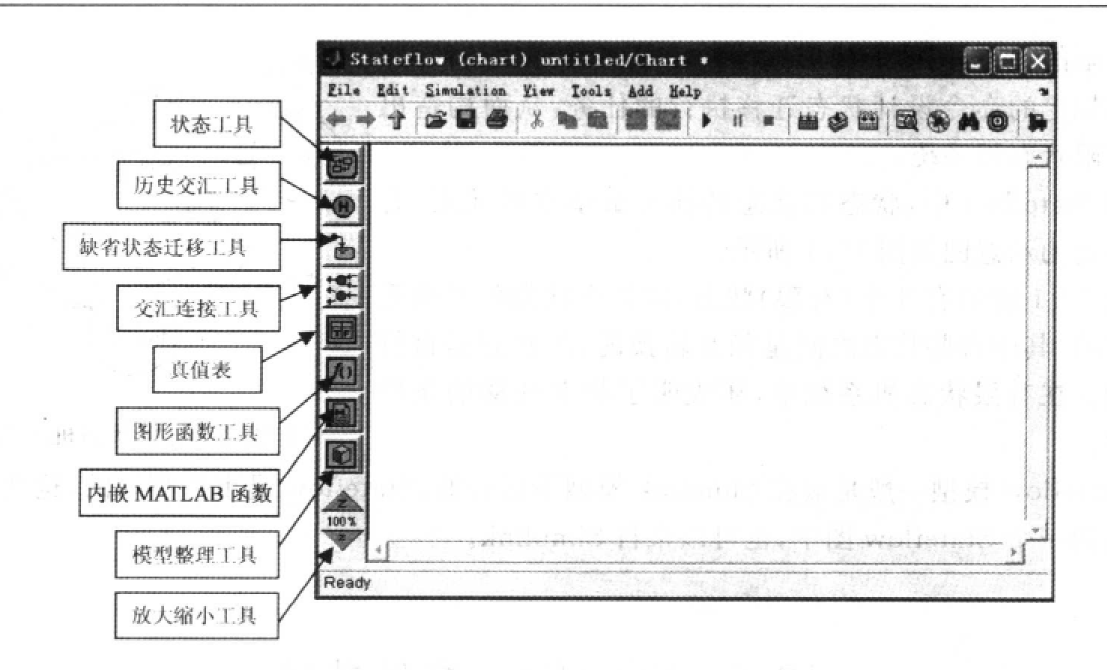


图 12.3　Stateflow 编辑界面

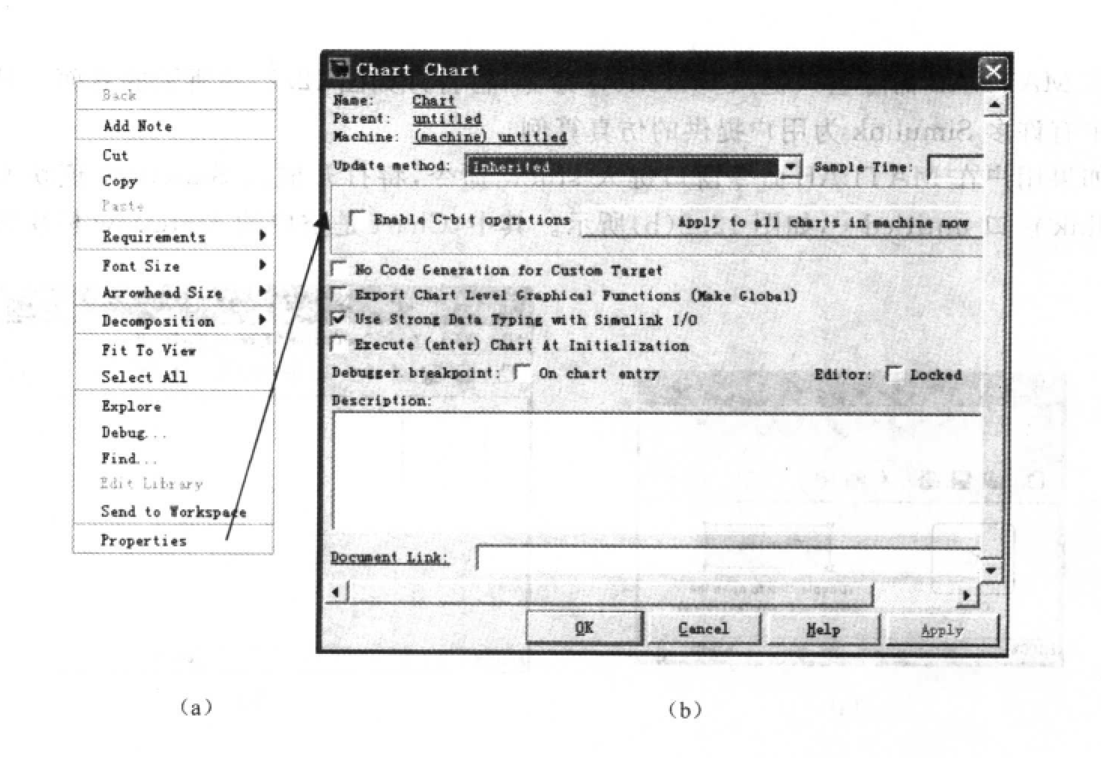


图 12.4　Stateflow 模型的属性设置

(a) Stateflow 快捷菜单；(b) Stateflow 属性设置对话框

用户可以利用 Stateflow 编辑界面左侧的各编辑工具绘制 Stateflow 图形。下面介绍常用的编辑工具。

1. 状态工具

系统的状态是指系统运行的模态。在 Stateflow 下，状态有两种行为：活动的(active)和非活动的(inactive)。单击状态工具按钮并拖动到编辑界面的空白处，即可绘制出一个状态的示意模块。用户可以在该模块左上角的问号位置填写状态的名称及动作描述，如标记为 On，本

例中状态的动作描述为 entry:speed = 1，将 speed 的值赋为 1（事实上，状态动作有很多种，不止 entry 一种，为了使读者先掌握状态流的一般知识，复杂状态流的状态动作类型及其应用将在随后的部分再阐述）。选中 On 模块，使用热键 Ctrl+C 和 Ctrl+V 或使用 edit 菜单下的复制及粘贴命令，即可再复制一个同样的模块，将复制的模块标记改为 Off，动作描述改为 entry:speed = 0。使用该工具，可以绘制出所有需要的状态，如图 12.5 所示。

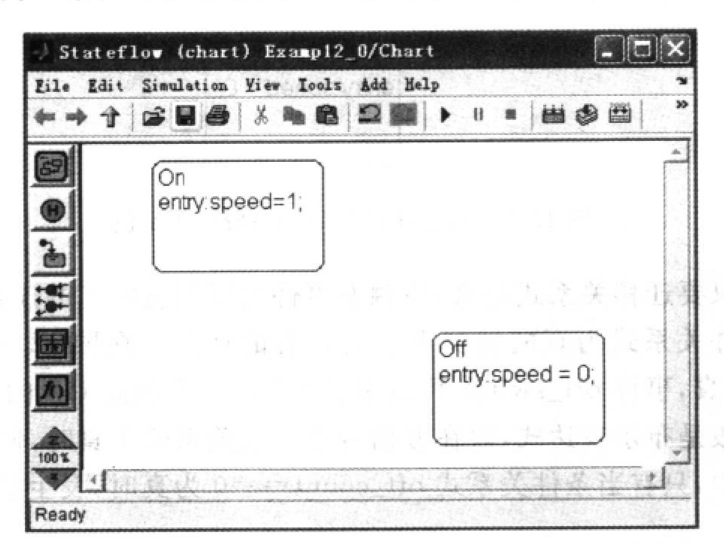


图 12.5　Stateflow 窗口的新建状态及其设置

如果右击建立的状态图标，并选择快捷菜单中的 Properties 菜单项，可打开图 12.6 所示的设置状态属性的对话框。用户也可以在 Label 栏填写状态的名称和动作描述。

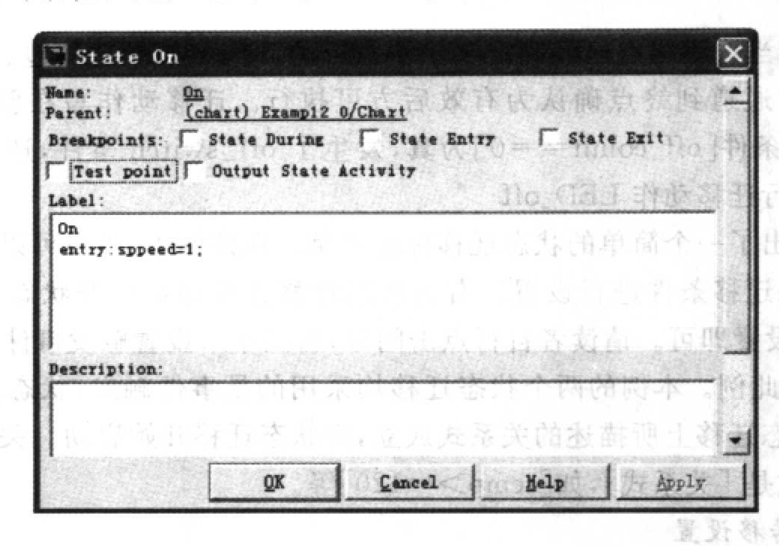


图 12.6　状态属性设置对话框

2. 状态迁移关系设置

在一个状态的边界按下鼠标键并拖动至另一个状态的边界释放，可以绘制出从一个状态到另一个状态的连线。单击此连线，在该连线上会出现一个问号，用户可以在该问号处添加状态迁移标记。状态迁移标记可以含有触发事件、迁移条件、条件动作及迁移动作，或它们中的任意组合。状态迁移标记的一般形式是

触发事件[迁移条件关系式]{条件动作}/迁移动作

图 12.7 中的状态迁移显示了状态迁移标记的一般形式的示例。

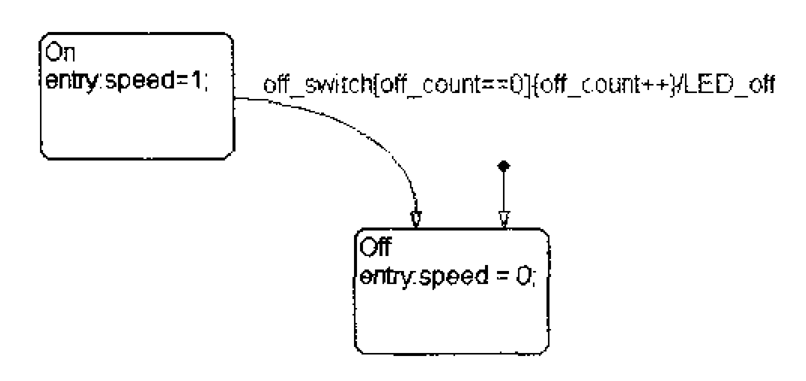


图 12.7 状态迁移标记的一般形式举例

触发事件表示只要迁移关系式是真，该触发事件可以引发状态的迁移。缺省触发事件时，任何事件均可在条件关系式为真的情况下引发状态的迁移。在图 12.7 的示例中，只有条件[off_count==0]为真，事件 off_switch 可以引发状态 On 至状态 Off 的迁移。

条件关系式一般是布尔表达式，写在方括号中。该关系式为真时，使得对于特定的触发信号迁移有效。本例中，只有当条件关系式 off_count==0 为真时，发生的事件 off_switch 才可引发状态迁移。

条件动作是指当条件关系式一旦成立(即为真)，就执行的动作，通常发生在迁移终点被确定有效之前。如果没有规定条件关系式，则认为条件关系式为真，即刻执行条件动作。条件动作必须写在花括号中。在图 12.7 的示例中，只要条件[off_count==0]为真，即可执行条件动作 off_count++。

迁移动作是指当迁移终点已经确定有效才执行的动作。如果迁移包含很多阶段，迁移动作只有在整个迁移通道到终点确认为有效后方可执行。迁移动作写在斜线"/"之后。在图 12.7 的示例中，当条件[off_count==0]为真，发生了 off_switch 事件，迁移终点状态 Off 确认为有效，此时执行迁移动作 LED_off

图 12.8 也给出了一个简单的状态迁移标记示例。在该例中，用户可以使用状态迁移属性设置对话框对状态迁移条件进行设置。右击状态迁移连线即可打开状态迁移属性设置对话框，在 Label 栏中设置即可。请读者自行点击问号，将图中未设置触发事件的迁移事件设置为 on_switch，并保存此例。本例的两个状态迁移均采用的是事件触发，状态迁移也可以采用关系式触发，一旦状态迁移上所描述的关系式成立，则状态迁移开始启动。关系式触发的状态迁移上的关系式格式是 [关系式]，如[temp>=120]等。

3. 缺省状态转移设置

缺省状态转移设置的作用是告诉 Stateflow 图形，当它开始工作时，哪个状态先处于激活状态。点击 Stateflow 图形编辑界面中的图标，然后将鼠标移动至需要设置的状态即可。如图 12.8 所示。

4. 事件与数据设置

前面为状态迁移规定了迁移触发事件的名称，也就是说状态的迁移仅在这些事件发生的时候才开始。为了利用这些事件触发，必须先定义这些事件。定义 on_ switch 和 off_switch 事件需要以下几步：

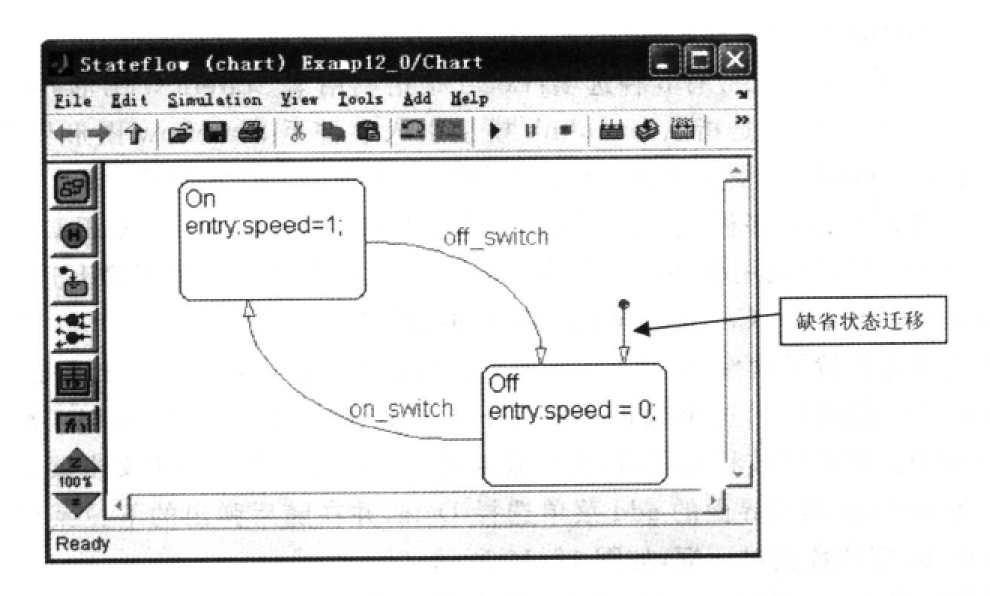


图 12.8　Stateflow 状态迁移设置

(1) 从 Stateflow 编辑界面的 add 菜单选择 Event，并在随后弹出的下拉菜单中选择 Input from Simulink，打开事件对话框，如图 12.9 所示；

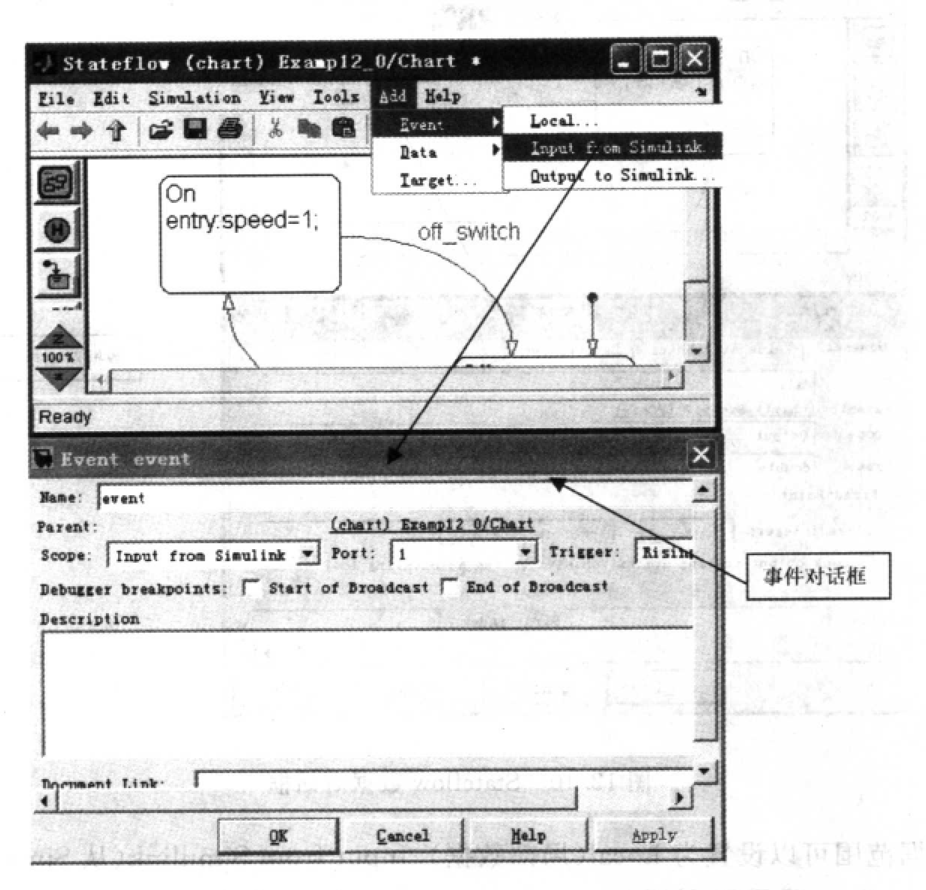


图 12.9　Stateflow 事件对话框

将事件对话框中的 Name 改为 off_switch，trigger 选择为 Falling(即下降沿触发)，点击 OK 保存 off_switch 事件的设置。

(2) 重复上述步骤设置 on_switch 事件，触发事件仍选择 Input from Simulink，Name 设

置为 on_switch，trigger 选择为 Rising（即上升沿触发）。

注意：事件的范围（Scope）有 3 种选项：Local 是指利用本 Stateflow 图形界面产生的触发事件；Input from Simulink 是指从 Simulink 模型引入事件至 Stateflow 图形界面；Output to Simulink 是指将 Stateflow 图形界面产生的事件输出到 Simulink 模型中。

事件的触发方式亦有多种选择：Either，Rising，Falling 和 Function Call 四种。其中选择 Rising 或 Falling 分别指利用事件的上升沿或下降沿触发，Either 是指不管上升沿还是下降沿事件均可以触发，Function Call 是一种函数调用的触发方式。

前面还为状态设置了动作，如状态 On 的动作描述为 entry：speed ＝ 1，是希望在状态 On 激活时将 speed 的值赋为 1，这个数据是要在 Simulink 模型中使用的，所以要将数据传递到 Simulink 模型中。在能够被利用之前，这个数据必须先定义。数据的定义步骤如下：

（1）从 Stateflow 编辑界面的 add 菜单选择 Data，并在随后弹出的下拉菜单下选择 Output to Simulink，打开数据对话框，如图 12.10 所示；

（2）将数据名 Name 改为 speed，点击 OK 保存设置即可。

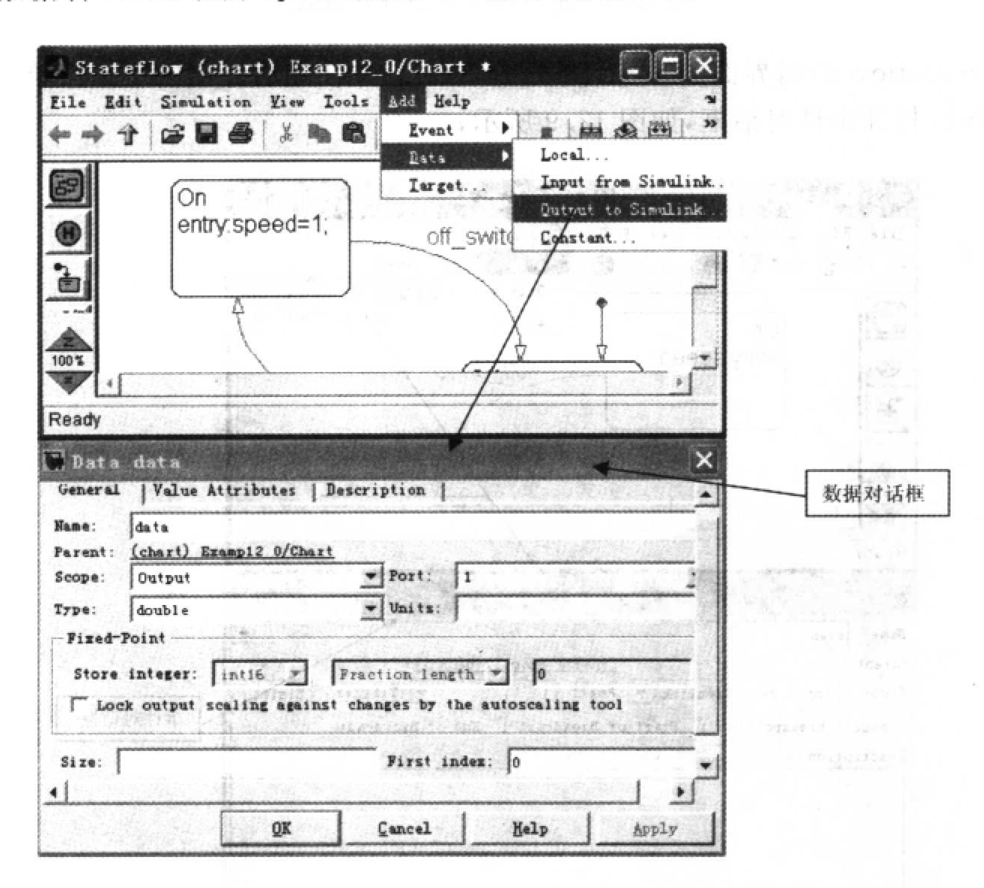


图 12.10　Stateflow 数据对话框

注意：数据范围可以设置为 Local（局部数据），Input from Simulink（从 Simulink 模型中输入数据，当 Stateflow 图需要利用 Simulink 模型的数据时，需要将相应的数据输入到 Stateflow 图中），Output to Simulink（向 Simulink 模型输出数据）和 Constant（常数）四种形式。数据的类型可以是 Double（双精度），Single（单精度），Int32（整数）及 Boolean（布尔数）等，也可以设置为 Inherited，即继承原来的设置。

这样在 Stateflow 编辑界面中，选择 Tools 菜单中的 Explore，将会打开模型管理器 Model

Explorer，如图 12.11 所示。

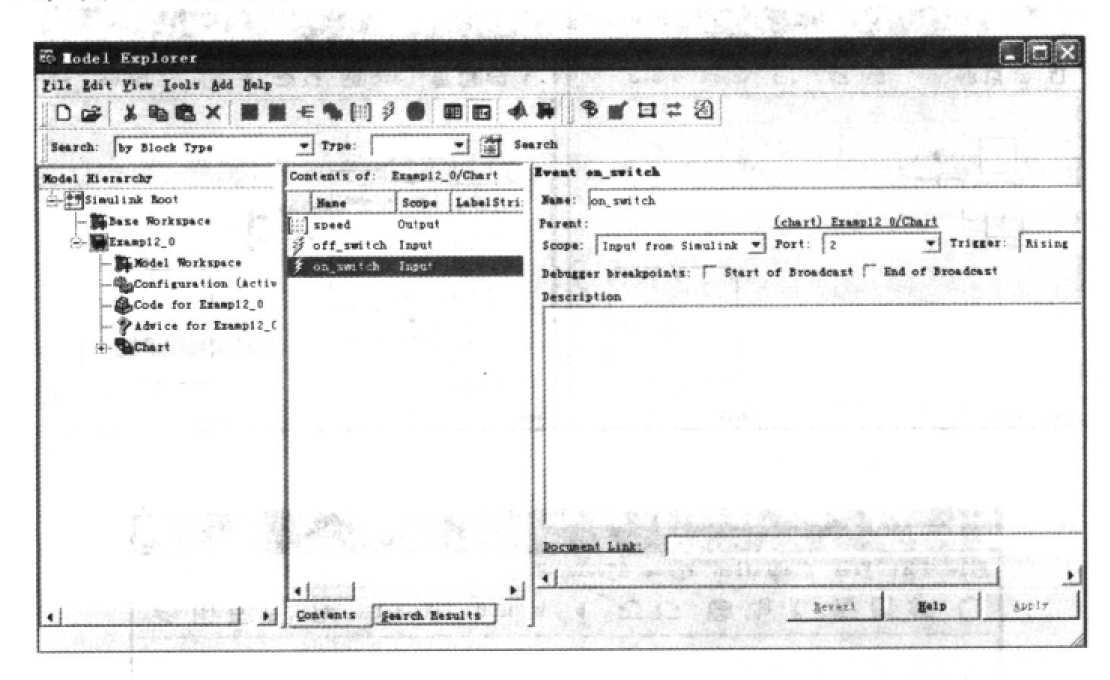


图 12.11　Model Explorer（模型管理器）

事件和数据的定义也均可以在模型管理器中进行。分别点击模型管理器工具栏上的图标 和 即可添加事件和数据，然后点击相应的内容修改事件的范围、触发方式或数据的范围及数据的类型等内容。

设置了事件和数据后，就可以从 Simulink 模型中输入事件和数据，并将 Stateflow 图执行过程中产生的数据或事件输出至 Simulink 模型中。

例 12.1　将前面叙述中已设置好数据和事件的 Stateflow 图存为 Examp12－1，从 Simulink中产生上升沿和下降沿的触发事件，将 Stateflow 图状态动作产生的数据 speed 输出到 Simulink 模型中并加以显示。

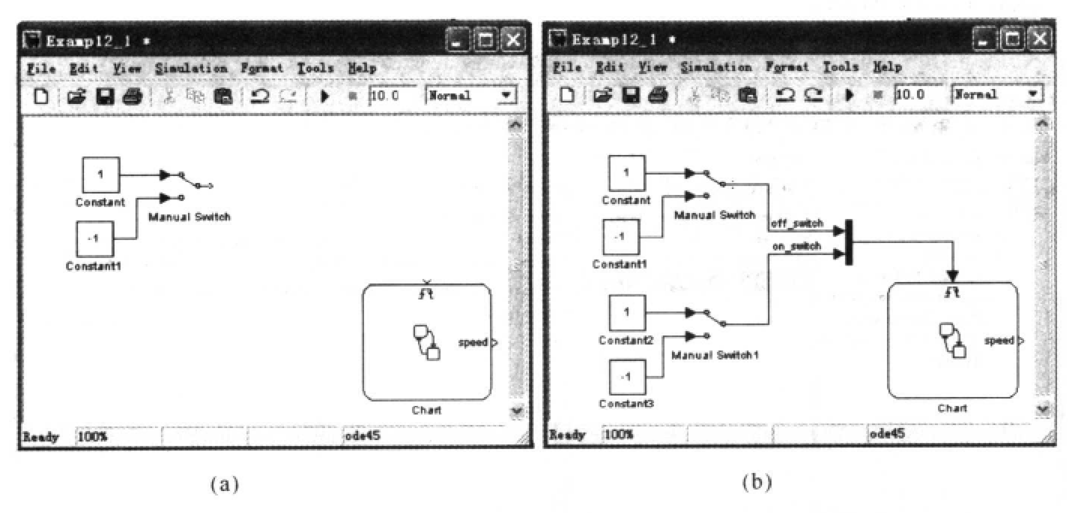
解　(1) 建模。前面的叙述已经讲到了 Stateflow 图的建立方法，包括状态模块的绘制及其设置、状态迁移及缺省状态迁移的设置、事件和数据的设置。下面在 Simulink 中产生具有上升沿或下降沿的触发事件 on_switch 和 off_switch，并将数据 speed 的值在 Simulink 模型中显示出来。

为了产生上升沿或下降沿的触发信号，需要使用 Simulink 模型库中的 Sources 库中的两个 Constant 模块及一个 Signal Routing 库中的 Manual Switch 模块，如图 12.12(a)所示。因为本例需要两个触发信号，而 Stateflow 模块的触发口仅有一个，所以必须使用 Signal Routing 库中的 Mux 模块组合信号，如图 12.12(b)所示。将 Sinks 模型库中的 Scope 模块拖进 Simulink 模型中以便显示 speed 的数值。最终的模型如图 12.12(c)所示。

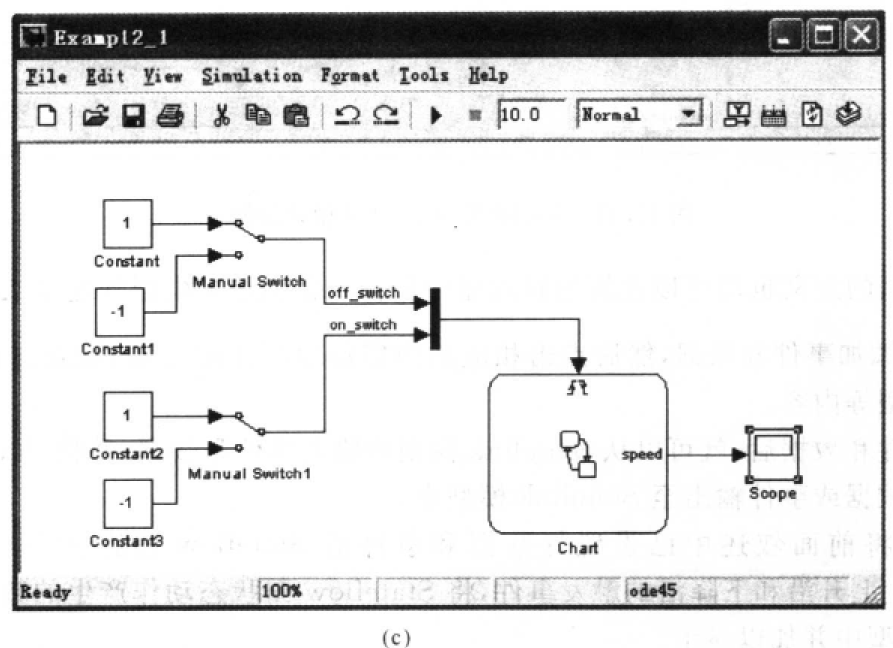
(2) 仿真。缺省情况下，可以直接对含有 Stateflow 模块的 Simulink 模型进行仿真。

1)像一般的 Simulink 模型仿真一样设置仿真参数：在 Simulink 模型窗口的 Simulation 菜单中选择 Configuration Parameters，打开仿真参数设置对话框，将 Stop time 设置为 inf，如图 12.13 所示。仿真结束时间设置为 inf 表示由用户自行决定仿真结束的时刻。

2)选择 simulation 菜单中的 start 或直接点击 Simulink 模型窗口上的图标▶启动仿真。



(a)　　　　　　　　(b)



(c)

图 12.12　例 12.1 的建模过程

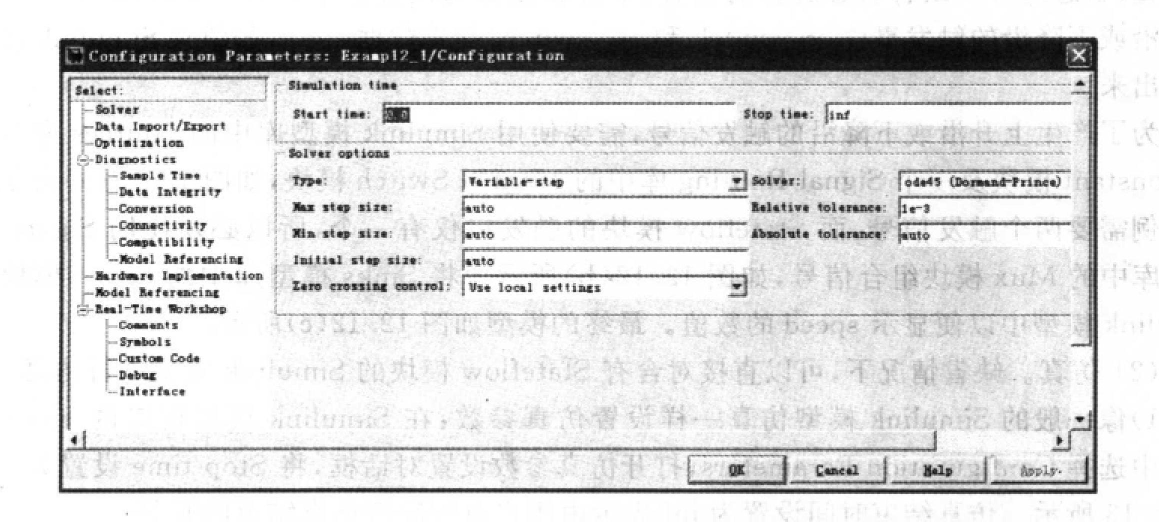


图 12.13　例 12.1 的仿真参数设置

注意：在启动仿真前，适当放置 Simulink 模型窗口、Stateflow 图形编辑界面和 Scope 示波器的位置以便既可以设置手动开关（Manual Switch）以产生所需的上升沿或下降沿触发信号，又可以同时观察 Stateflow 图形窗口中状态迁移情况及 Scope 模块中数据显示的变化情况。如图 12.14(a)所示，并将手动开关置于图示位置。

启动仿真后，Stateflow 先分析 Stateflow Chart 图是否存在错误，如果有错误，会弹出一个窗口告诉用户错误的类型及位置；如果没有错误，则 Stateflow 为每个 Stateflow Chart 图自动生成 S-函数（Mex 文件），并将生成的 S-函数放到 MATLAB 当前目录中的 sfprj 子目录下（如果 MATLAB 当前目录中没有 sfprj 子目录，MATLAB 会自动建立此目录）。仿真时，Simulink 只需调用这些 S-函数。

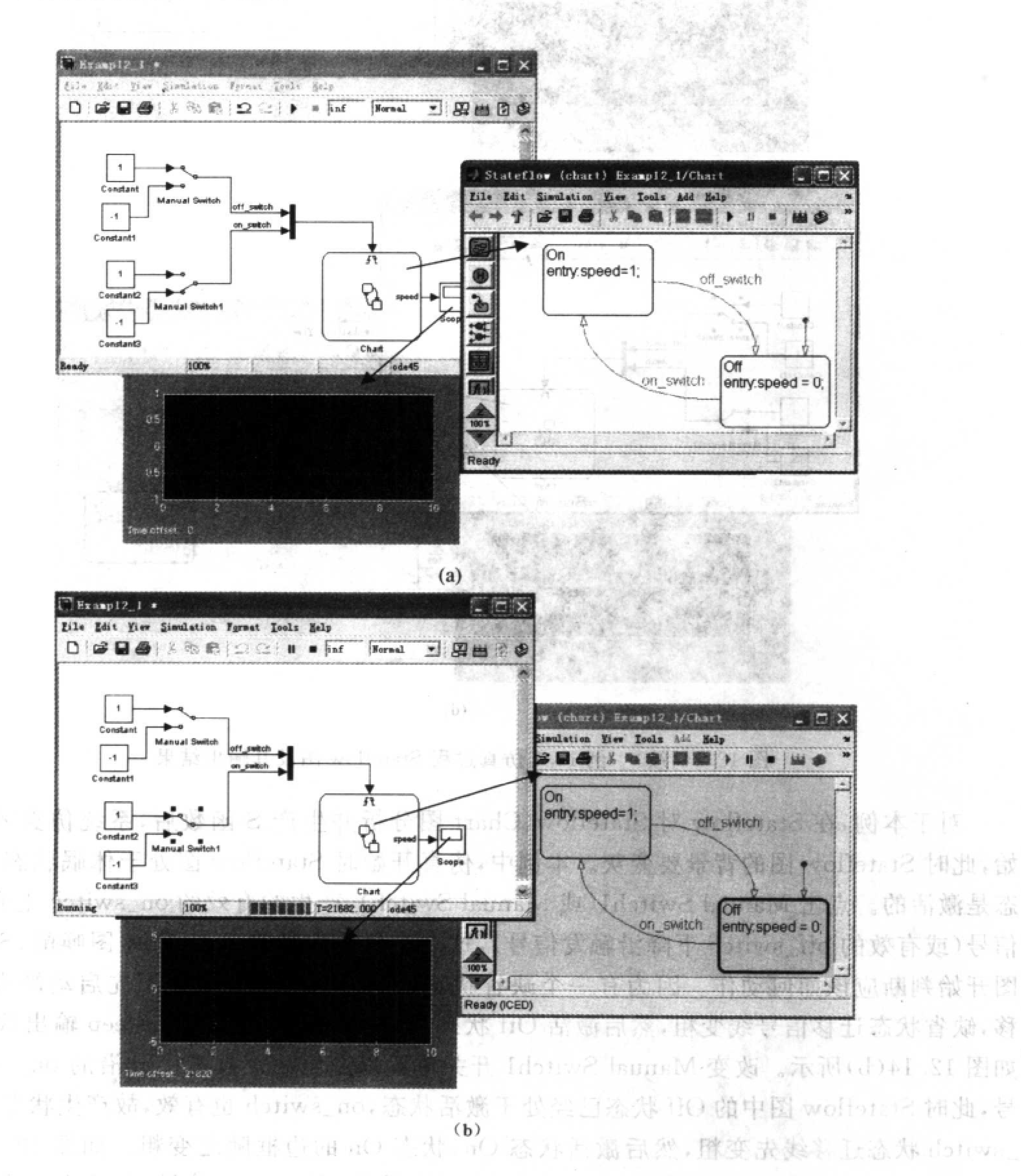


图 12.14　例 12.1 仿真过程 Stateflow 图及其输出结果

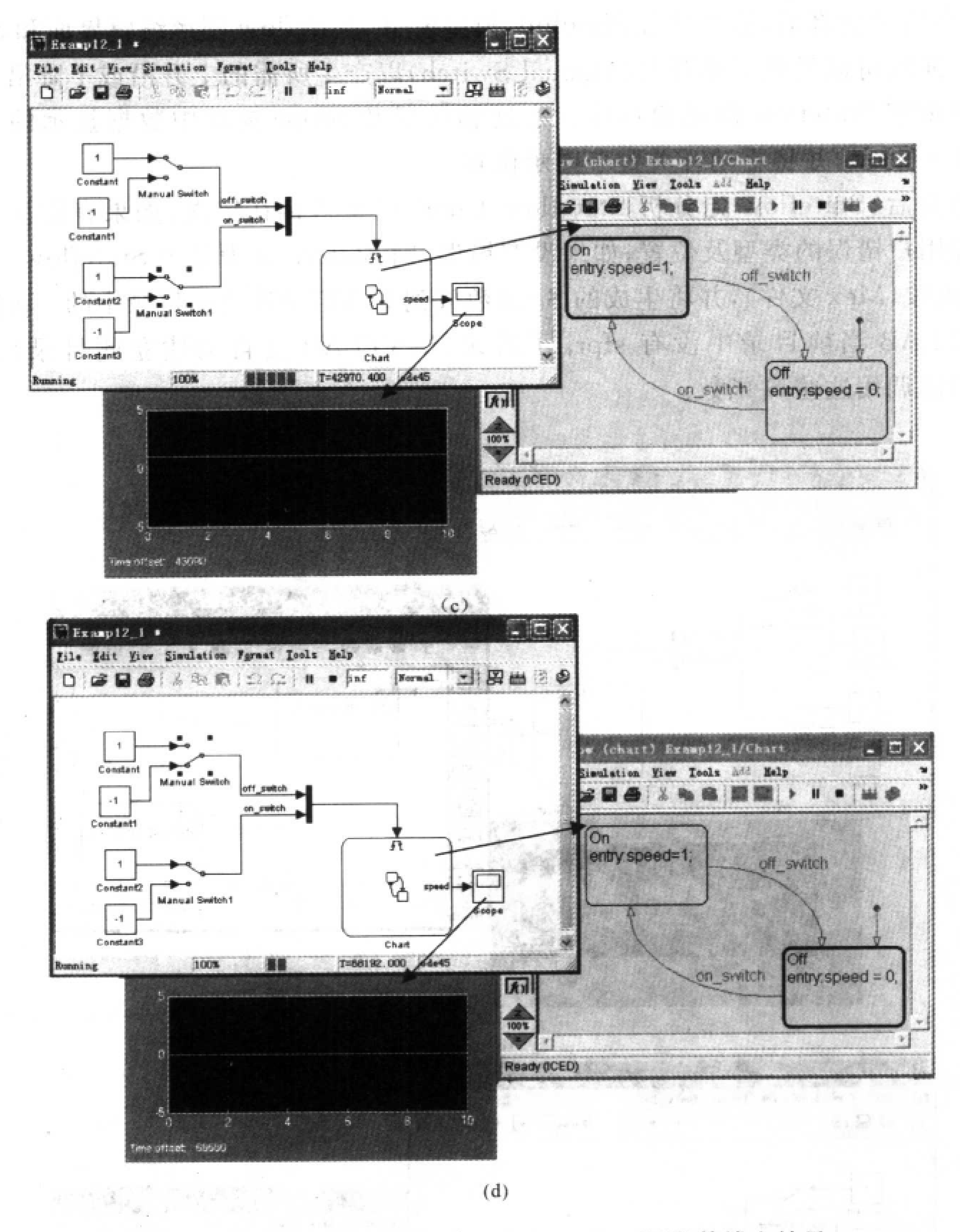


图 12.14(续)　例 12.1 仿真过程 Stateflow 图及其输出结果

对于本例，在 Stateflow 对 Stateflow Chart 图分析并生产 S-函数后，系统仿真才真正开始，此时 Stateflow 图的背景要变灰。本例中，仿真开始时 Stateflow 图处于休眠状态，没有状态是激活的。点击 Manual Switch1(或 Manual Switch)，产生的有效的 on_switch 上升沿触发信号(或有效的 off_switch 下降沿触发信号)，这个有效的信号将 Stateflow 图唤醒，Stateflow 图开始判断应该如何动作。因为有一个缺省状态迁移，所以 Stateflow 图先启动缺省状态迁移，缺省状态迁移信号线变粗，然后激活 Off 状态，Off 状态的边框变粗，sleep 输出数值是 0，如图 12.14(b)所示。改变 Manual Switch1 开关位置以产生一个具有上升沿的 on_switch 信号，此时 Stateflow 图中的 Off 状态已经处于激活状态，on_switch 也有效，故产生状态迁移，on_switch 状态迁移线先变粗，然后激活状态 On，状态 On 的边框随之变粗。如图 12.14(c)所示。改变 Manual Switch 开关位置以产生一个下降沿的 off_switch 信号，此时 Stateflow 图中的 On 状态是激活状态，off_switch 触发信号有效，将产生从状态 On 到状态 Off 的迁移，off_

switch 状态迁移线先变粗，然后激活状态 Off，状态 Off 的边框随之变粗。如图 12.14(d) 所示。

5. 交汇连接设置

使用 Stateflow 编辑界面中的交汇连接工具可以产生交汇连接点。交汇连接点主要用来处理状态迁移过程中的迁移信号的分离和汇合。用户只需激活交汇连接工具，然后将鼠标移动适当的位置单击鼠标即可设置一个交汇连接点。我们通过下面的例子讲述交汇连接工具如何进行信号的汇合及分离的。

例 12.2　在例 12.1 的基础上修改系统模型，给系统增加一个温度传感器，使得其每0.5 s 测量一次温度，系统需要根据温度数值的大小确定输出 speed 值。这种情况下，Simulink 模型需要给 Stateflow 模块输入数据 temp，Stateflow 图中需要根据 temp 值的大小判断需要激活哪个状态。

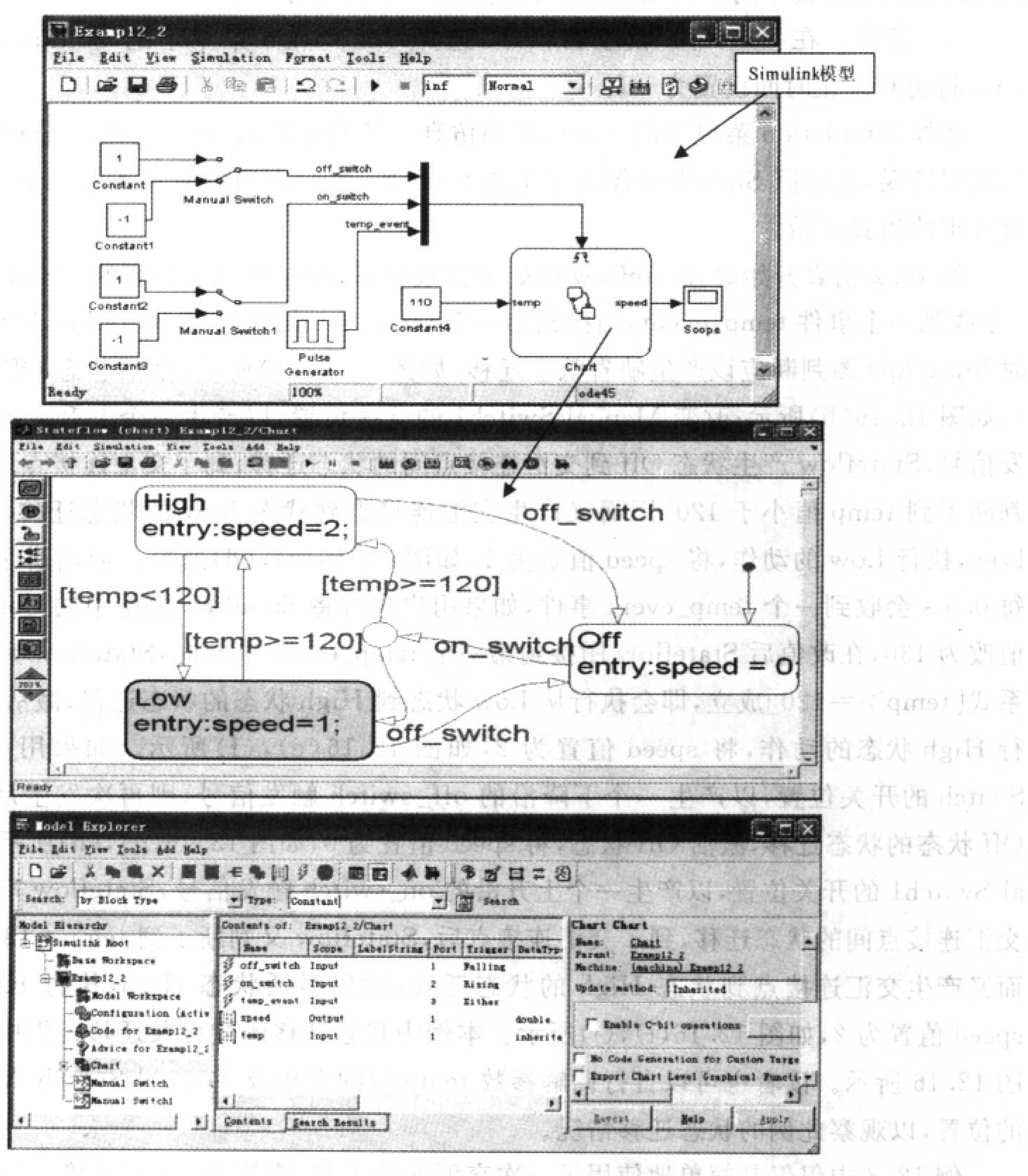


图 12.15　例 12.2 的 Simulink 模型、Stateflow 图及其模型管理器的事件、数值设置

解 (1) 建模。在例 12.1 的基础上,修改系统模型如图 12.15 所示。

在 Simulink 模型中,用 Constant4 模块模拟测量的温度值,开始时设置为 110。Pulse Generator 产生的周期和幅值均为 1 的方波信号用来触发 Stateflow 图,每 0.5 s 判断一次温度值的大小。实际上,Pulse Generator 产生的方波信号可以模拟温度测量的采样周期。

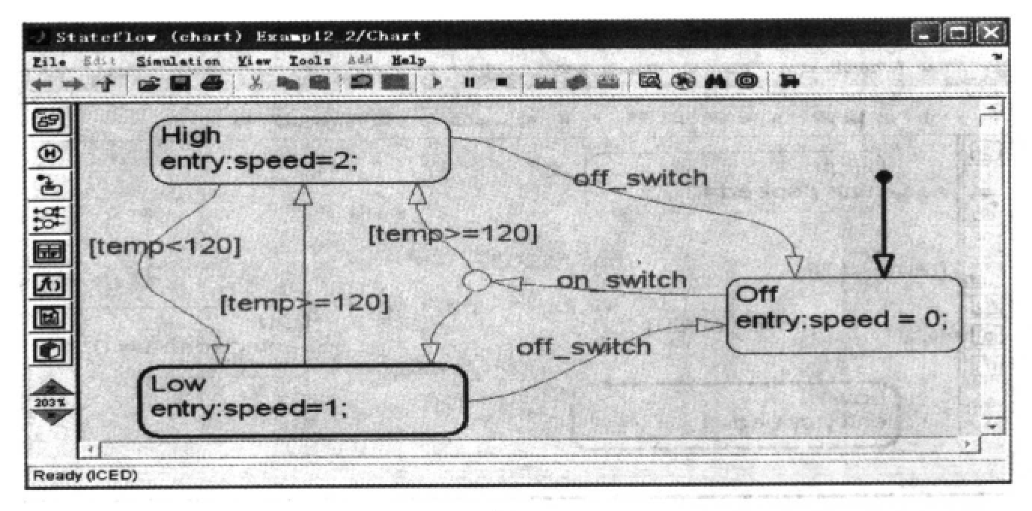
请读者按照图 12.15 所示建立例 12.2 的 Stateflow 图。此 Stateflow 图含有一个交汇连接工具。此 Stateflow 图的数值 temp 是需要从 Simulink 模型中输入的,除此之外,还需利用来自 Simulink 模型中的 temp_event 事件以触发此 Stareflow 图每 0.5 s 读一次 temp 数值并判断是否应该状态迁移。因而在例 12.1 的基础上,系统中还需添加一个 Input from Simulink 的事件 temp_event 和一个 Input from Simulink 的数值 temp。将事件 temp_event 的触发沿设置成 either。添加后,选择 Stateflow 图中的 Tools 菜单下的 Explorer,打开模型管理器 Model Explore,其中的事件和数值设置应如图 12.15 所示。

(2) 仿真。在 Simulink 模型窗口中,选择 Simulation 菜单下的 Configuration Parameters,将仿真终止时间设置为 inf。

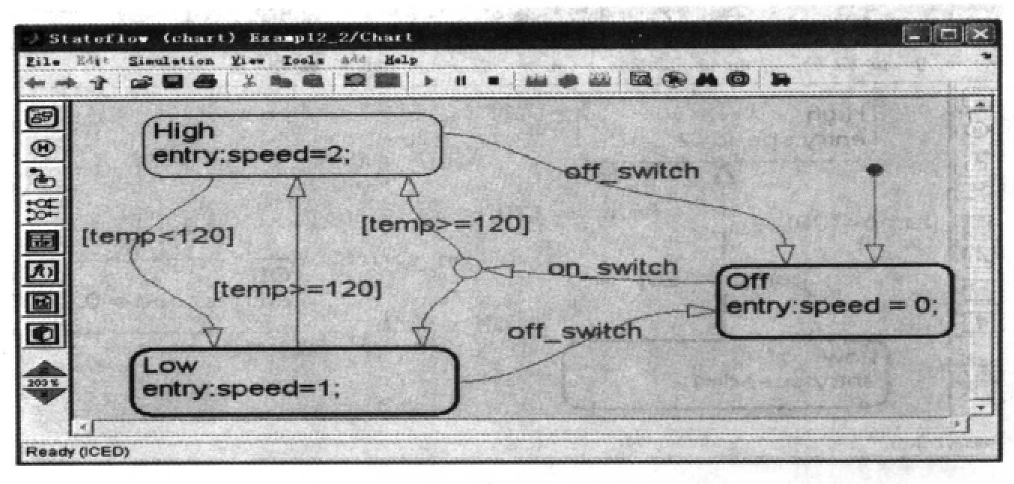
选择 Simulation 菜单下的 Start,启动仿真。仿真开始时,Simulink 先分析 Stateflow 图,如没有错误,就按照 Stateflow 图自动生成 S-函数,并存在当前目录下的 sfprj 子目录下。这时真正的仿真开始。

例 12.2 仿真开始时,Stateflow 图处于休眠状态,没有状态是激活的。Stateflow 图每 0.5 s 会收到一个事件 temp_event,当接到第一个 temp_event 事件时,Stateflow 图就被唤醒了,此时 Stateflow 图判断应该产生缺省状态迁移,如图 12.16(a)所示;激活状态 Off,speed 输出为 0,如图 12.16(b)所示;改变 Manual Switch1 的开关位置,以产生一个上升沿的 on_switch 触发信号,Stateflow 产生状态 Off 到交汇连接点间的状态迁移,到了交汇连接点后,Stateflow 又判断了到 temp 值小于 120,因而又产生交汇连接点到状态 Low 的状态迁移,此后激活状态 Low,执行 Low 的动作,将 speed 值置为 1,如图 12.16(c),(d)所示。前面讲了,Stateflow 图每 0.5 s 会收到一个 temp_event 事件,如果用户这时将 Simulink 模型中的 Constant4 模块的值改为 130,在改值后 Stateflow 图接到第一个 temp_event 事件时,Stateflow 判断条件转移关系式[temp＞＝120]成立,即会执行从 Low 状态到 High 状态的状态迁移,激活 High 状态,执行 High 状态的动作,将 speed 值置为 2,如图 12.16(e),(f)所示。如果用户改变 Manual Switch 的开关位置,以产生一个下降沿的 off_switch 触发信号,则再次发生从 High 状态至 Off 状态的状态迁移,激活 Off 状态,将 speed 值置为 0,如图 12.16(g),(h)所示。改变 Manual Switch1 的开关位置,以产生一个上升沿的 on_switch 触发信号,Stateflow 产生状态 Off 到交汇连接点间的状态迁移,到了交汇连接点后,Stateflow 又判断了到 temp＝130,大于 120,因而又产生交汇连接点到状态 High 的状态迁移,此后激活状态 High,执行 High 的动作,将 speed 值置为 2,如图 12.16(i),(j)所示。本例中状态迁移的过程及迁移过程中的信号流向详图 12.16 所示。读者也可以自行调解参数 temp 值的大小及 Manual Switch,Manual Switch1 的位置,以观察此例的状态迁移情况。

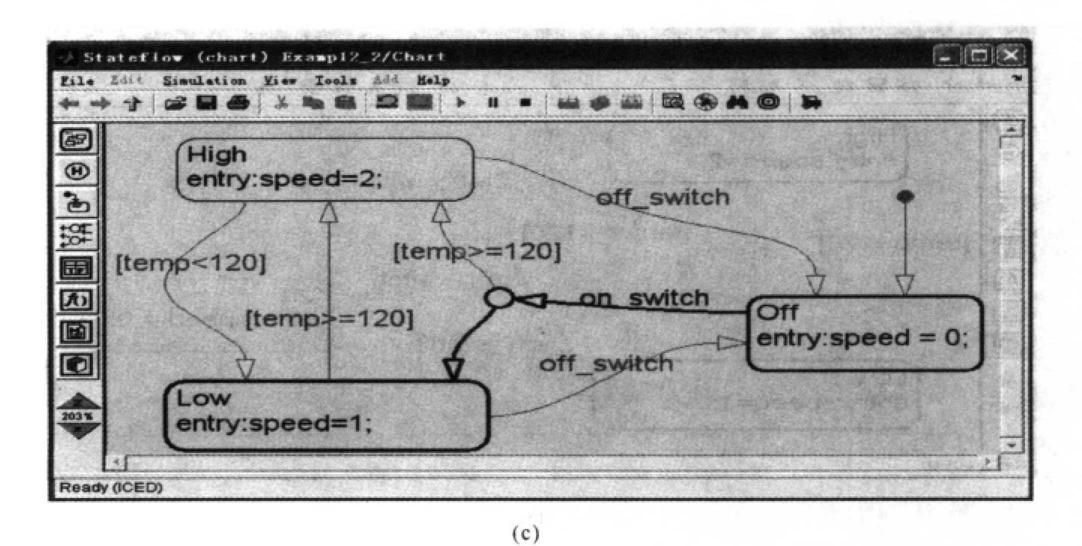
例 12.2 中仅仅是简单地使用了一次交汇连接工具,事实上,交汇连接工具的合理使用可以完成非常复杂的逻辑关系。如图 12.17(a)所示,实现如下 If_then 判断功能:



(a)

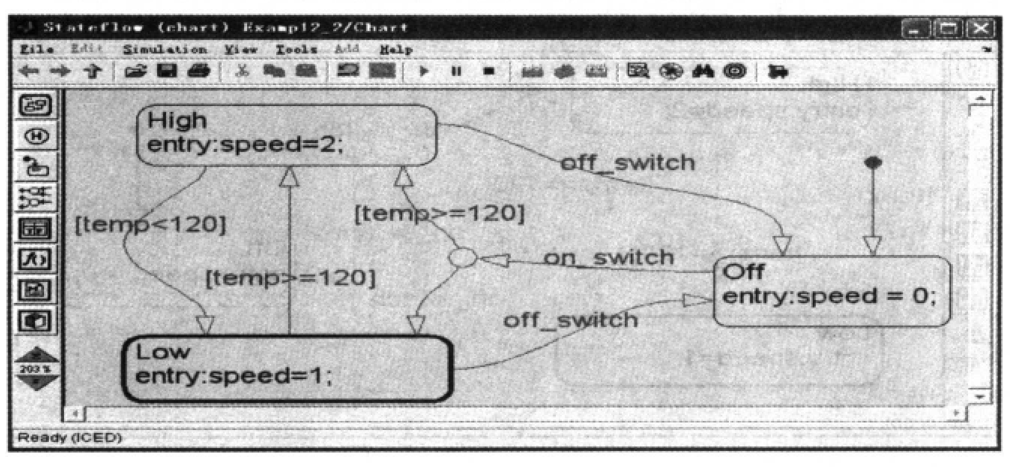


(b)

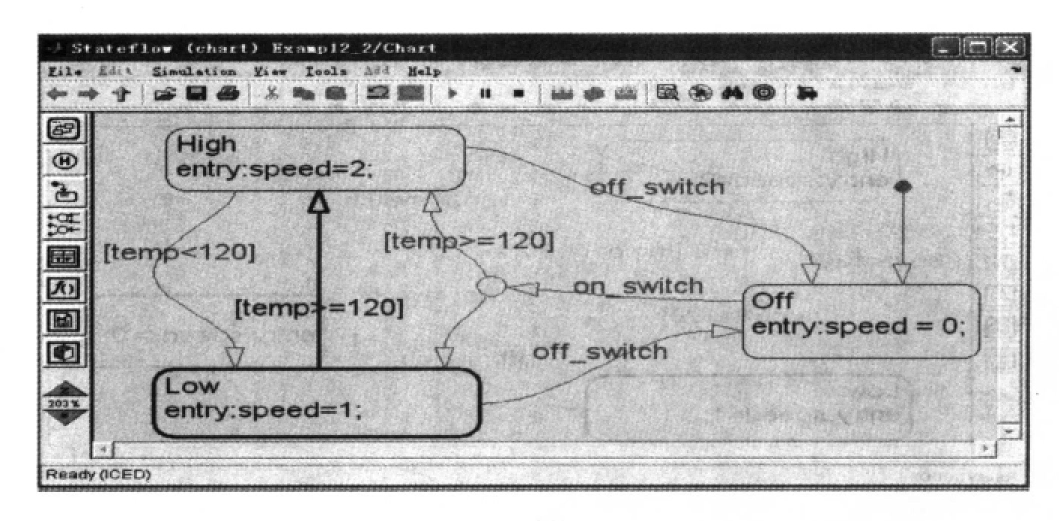


(c)

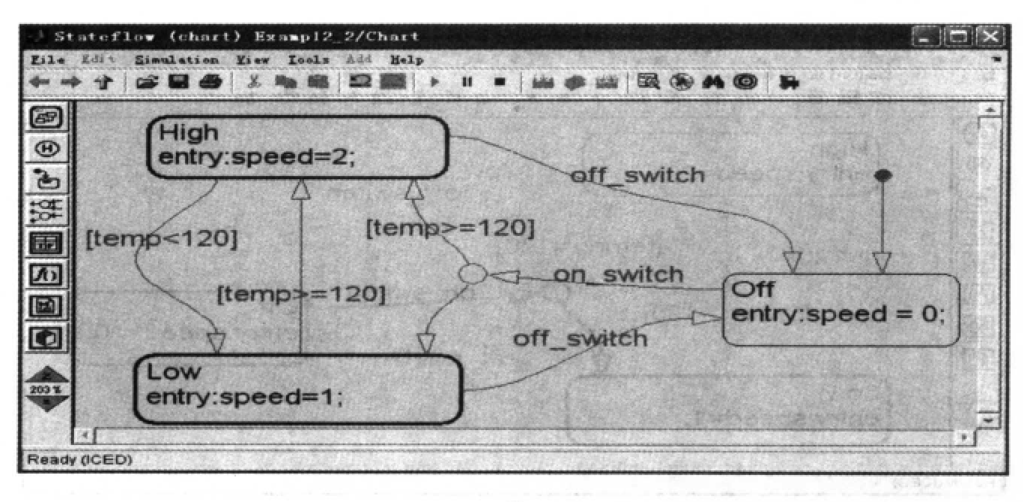
图 12.16　例 12.2 仿真时的状态迁移过程示意图



(d)

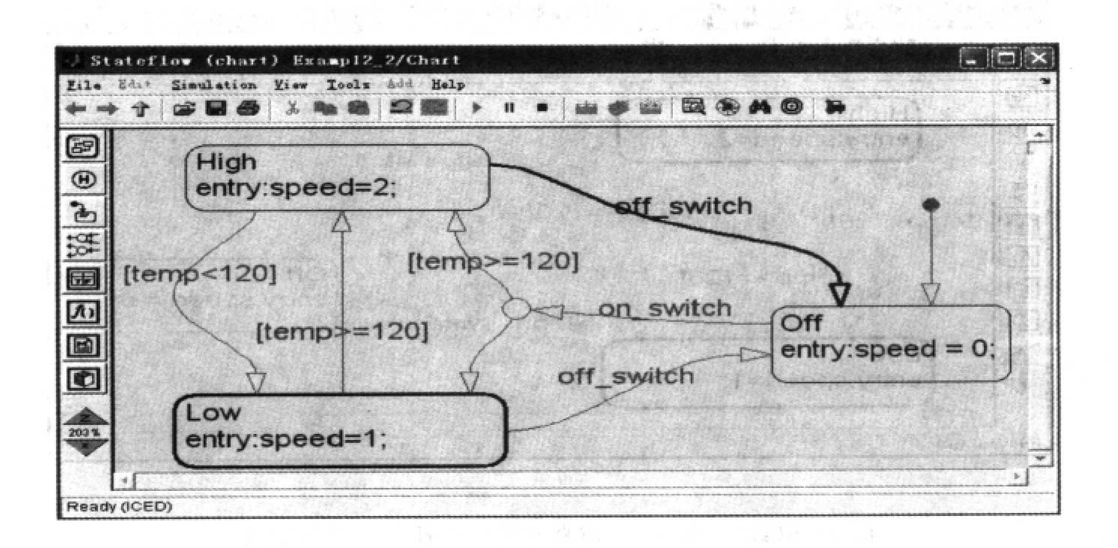


(e)

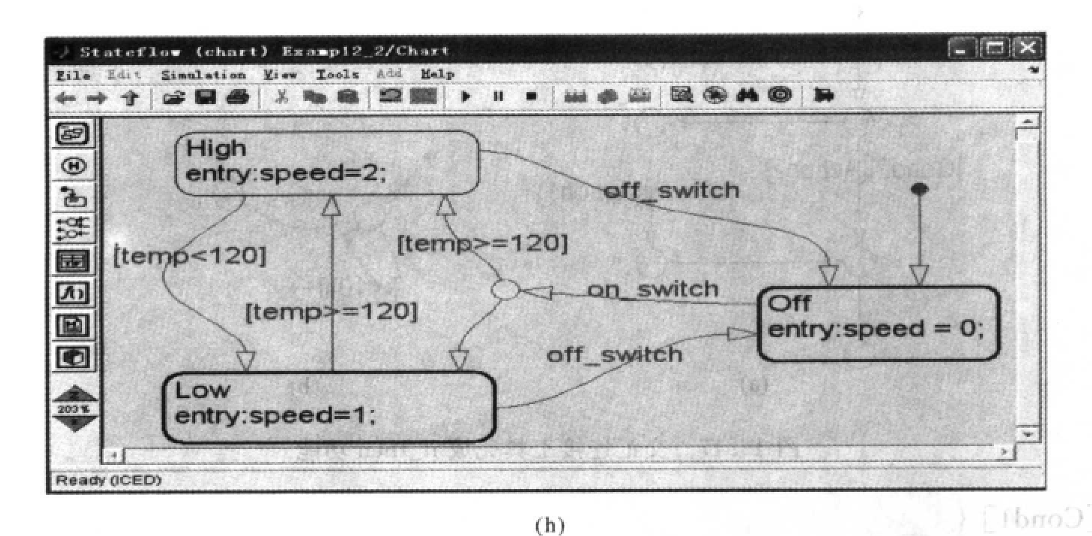


(f)

图 12.16(续)　例 12.2 仿真时的状态迁移过程示意图



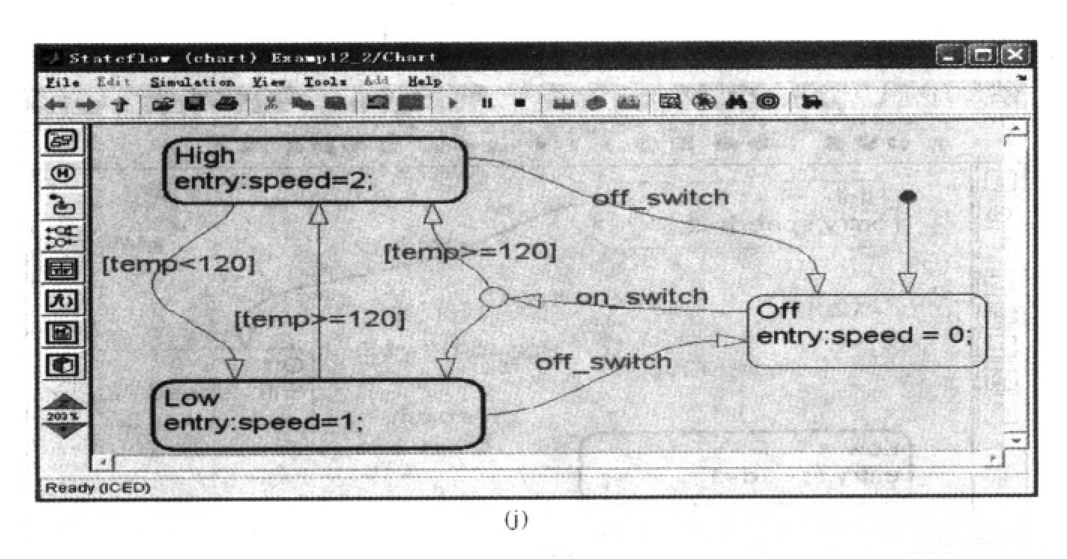
(g)



(h)

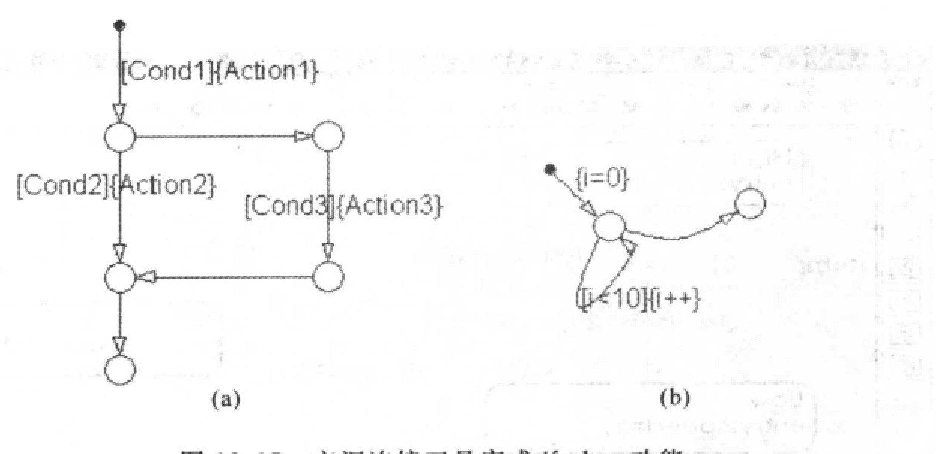
(i)

图 12.16(续)　例 12.2 仿真时的状态迁移过程示意图



(j)

图 12.6(续) 例 12.2 仿真时的状态迁移过程示意图



(a) (b)

图 12.17 交汇连接工具完成 if_then 功能

```
if [Cond1] {
Action1
if [Cond2] {

Action2
}elseif [Cond3]{
Action3
}
}
```

利用交汇连接工具除了可以实现 if_then 的判断功能，还可实现 For 循环功能等，见图 12.17(b)。

6. 图形函数的设置及其调用

例 12.2 的 Stateflow 图中多次利用了条件关系式[temp>=120]。对于这种多次重复使用的关系式，可以设置一个图形函数 Function，使用时调用这个函数即可。

状态流的图形函数是使用交汇连接工具和状态迁移工具绘制的状态流图形。用户可以建

立一个图形函数，在里面加入流程图，然后在状态的动作和迁移过程中反复调用。因为调用函数时，函数必须执行完全，所以图形函数中不能含有状态。一个最小的图形函数至少要包含一个缺省状态迁移和一个终止的交汇连接工具。

要在一个 Stateflow 图添加一个图形函数 Function，只需点击 Stateflow 图中的图形函数工具 f()，移动鼠标至 Stateflow 图中的适当位置，再点击鼠标左键即可。在图形函数 function 后写入函数的返回变量及函数名，格式为：返回形参＝函数名（形参）；回车后即可建立该图形函数。一旦建立了图形函数，用户可以在状态流的状态动作和状态迁移中反复调用它，调用的格式与函数的格式完全相同，只是需要将形参换成实际使用的参数变量。

例 12.3　将例 12.2 中的条件关系式写成图形函数，完成例 12.2 所有的状态迁移功能。

解　系统的 Simulink 模型不会发生变化，同例 12.2 的 Simulink 模型。

Stateflow 图需要引入图形函数。将例 12.2 另存为 exampl2_3，打开 Stateflow 图，点击左边的图形函数工具 f()，移动鼠标至合适的位置，点击鼠标左键。在 Function 后光标的位置键入 r=hot()，回车，再点击 Stateflow 图左边的缺省状态迁移工具，将鼠标移入图形函数点击鼠标左键，产生一个缺省状态迁移线，在此迁移线上的问号处输入函数关系式，本例为{r=temp>=120}，意思是将当 temp>=120 为真时置 r 为 1，否则 r 为 0。这样图形函数就设置好了。在调用这个图形函数时，只需在调用的位置写入函数名即可。例 12.3 的 Stateflow 图如图 12.18 所示。

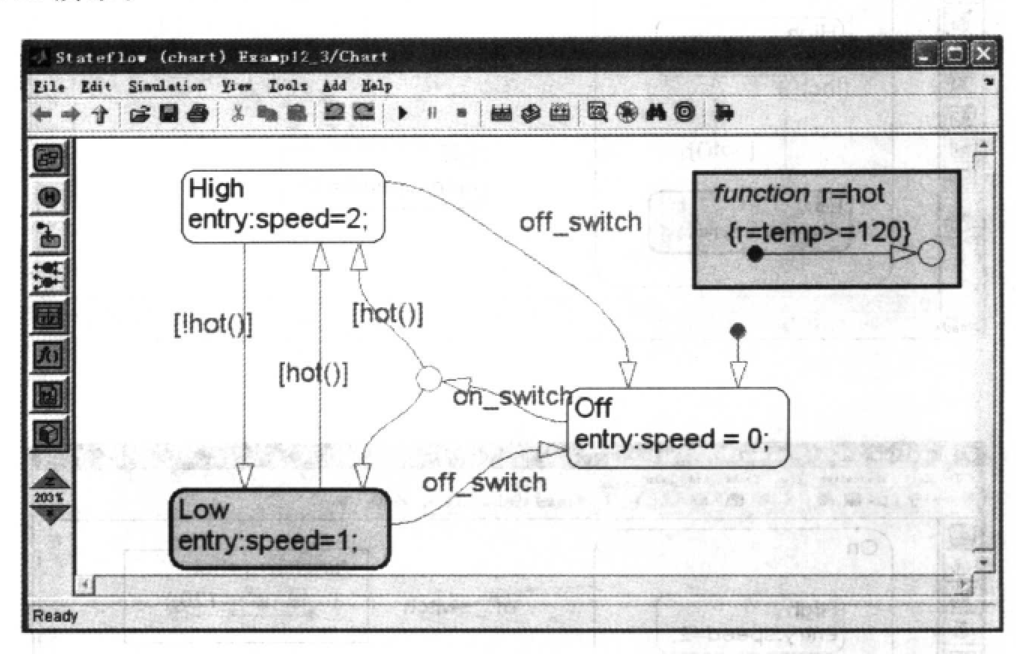


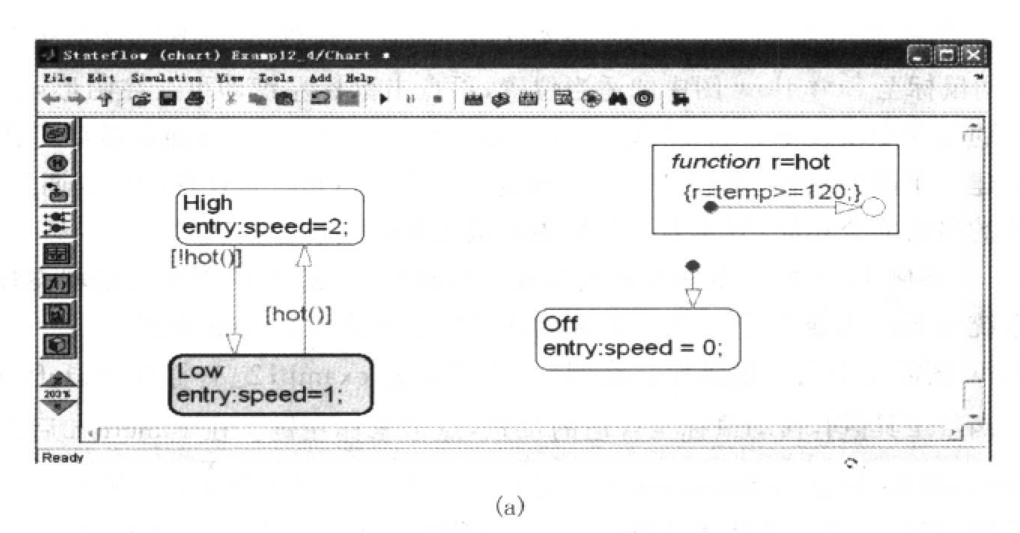
图 12.18　含图形函数的例 12.3 的 Stateflow 图

例 12.3 在仿真时状态的迁移情况类似例 12.2，只是例 12.3 在进行仿真计算时，图形函数中的状态迁移线常常处于激活状态（表现为状态迁移线常处于变粗的状态），这是因为 Stateflow 每 0.5 s 时要从 Simulink 处接受一次 temp 值，随后 Stateflow 需要运行此图形函数判断 temp 值是否大于 120，并给 r 赋值。

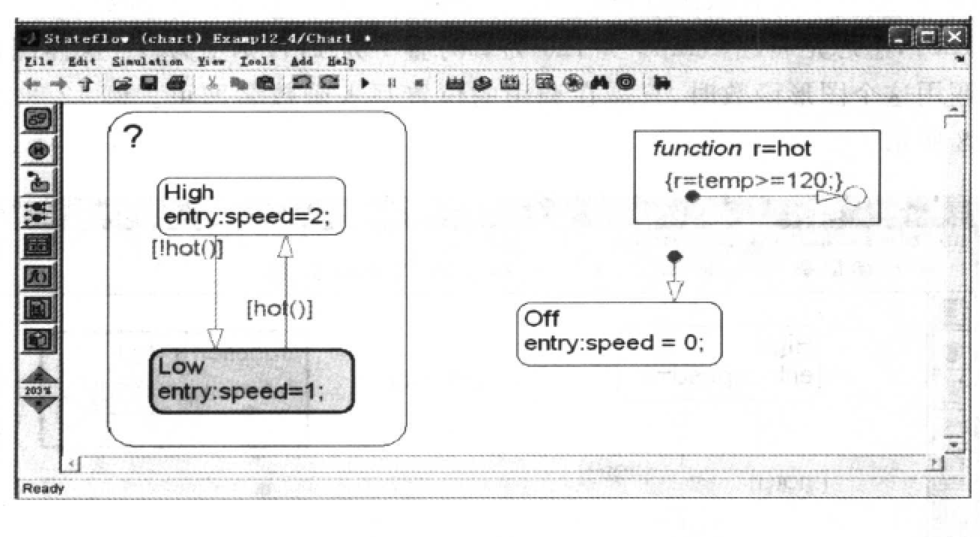
7. *多层状态的嵌套*

在例 12.3 中，当系统开始处于休眠状态时，状态 Off 激活时，状态 High 和 Low 实际上均

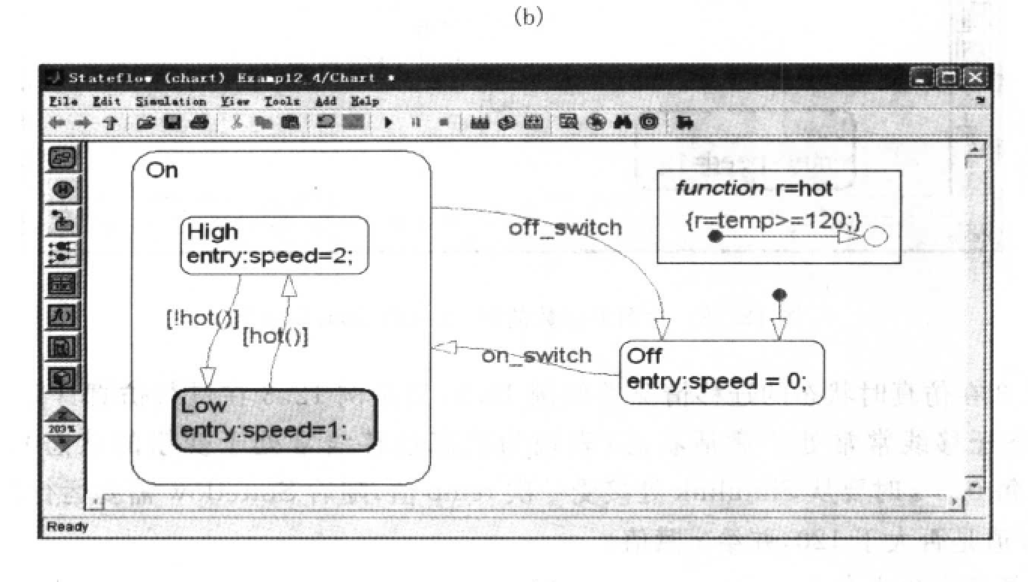
需要 On_switch 事件激活，这样可以利用一个状态来包含这两个状态以使 Stateflow 图更简单明了。这里利用例 12.4 说明如何进行多层状态的设计以及多层状态在仿真时如何进行状态迁移的。



(a)

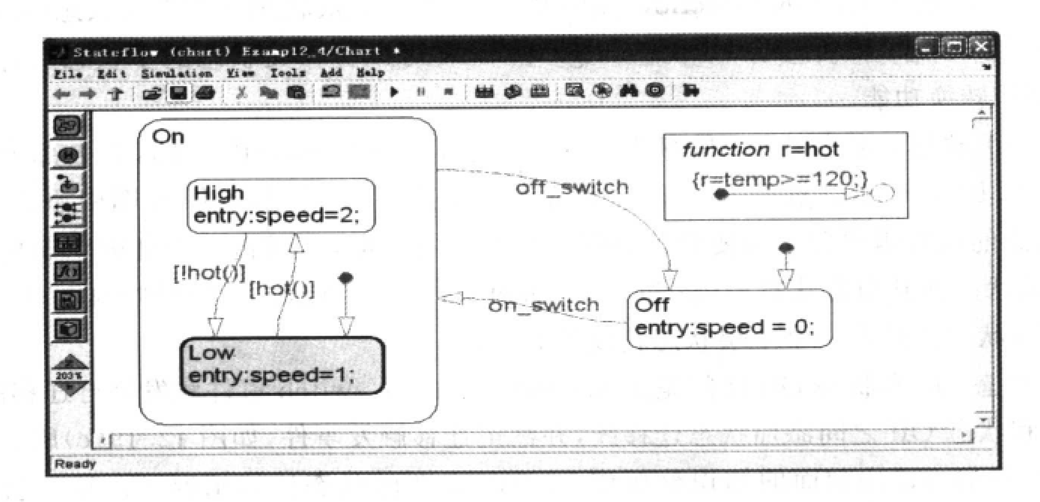


(b)

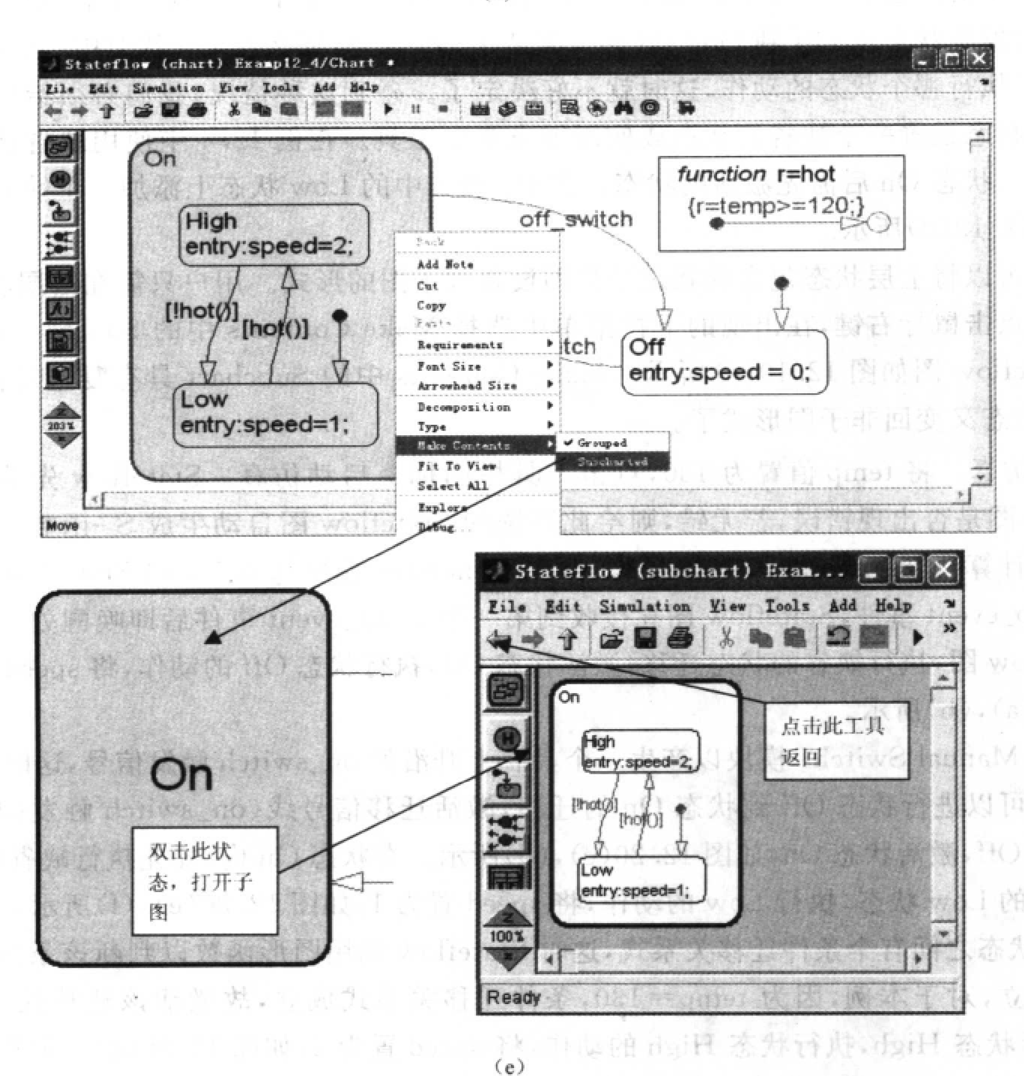


(c)

图 12.19 例 12.4 的 Stateflow 图设置过程



(d)



(e)

图 12.19(续)　例 12.4 的 Stateflow 图设置过程

例 12.4　在例 12.3 的 Stateflow 图的基础上，建立上层状态或称作父状态(superstate)，使其包含状态 High 和状态 Low(状态 High 和 Low 可以称作父状态的子状态)，并完成与例 12.3 类似的转换功能。

解　(1) 建模。将例 12.3 另存为 exampl2_4，打开其 Stateflow 图，删去部分状态迁移线，如图 12.19(a)所示。点击 Stateflow 图左侧的状态工具，在 Stateflow 图中添加一个状态，注意该状态的位置需要置于需要包含的两个状态的右下部，然后拖动新加的状态的左上角边框，增大该状态使其包含状态 High 和 Low，如图 12.19(b)所示。在问号处输入 On，即将此状态称作 On 状态。这样一个上层状态就设置好了。

那么状态 On 和状态 Off 之间是由 off_switch 和 on_switch 事件触发进行迁移的，我们在状态 On 和状态 Off 之间添加状态迁移线，并标记迁移触发事件，如图 12.19(c)所示。

至此，我们可以用前面的知识分析该 Stateflow 图的状态迁移情况了。这里会碰到一个问题：在 Off 状态处于激活状态时，Stateflow 图从 Simulink 模型中收到一个有效的 on_switch 信号，这时发生状态从 Off 到 On 的转移。激活了 On 状态，那么在 On 状态中，具体应该激活哪个状态，执行哪个状态的动作，这时就不好决定了。碰到这种情况，我们必须给这个上层状态包含的状态设置一个缺省的状态或使用历史交汇工具。在例 12.4 中使用缺省状态迁移工具指定进入状态 On 后需先激活的状态。在 On 状态中的 Low 状态上添加一个缺省状态迁移线，如图 12.19(d)所示。

用户可以将上层状态包含的状态迁移图设置成子图的形式。用户只需在上层状态 On 内的任意点点击鼠标右键，在出现的下拉菜单中选择 Make Contents 中的 Subchart 即可，设置后的 Stateflow 图如图 12.19(e)所示。Make Contents 中的 Subchart 具有复选功能，再选它时，上层状态又变回非子图形式了。

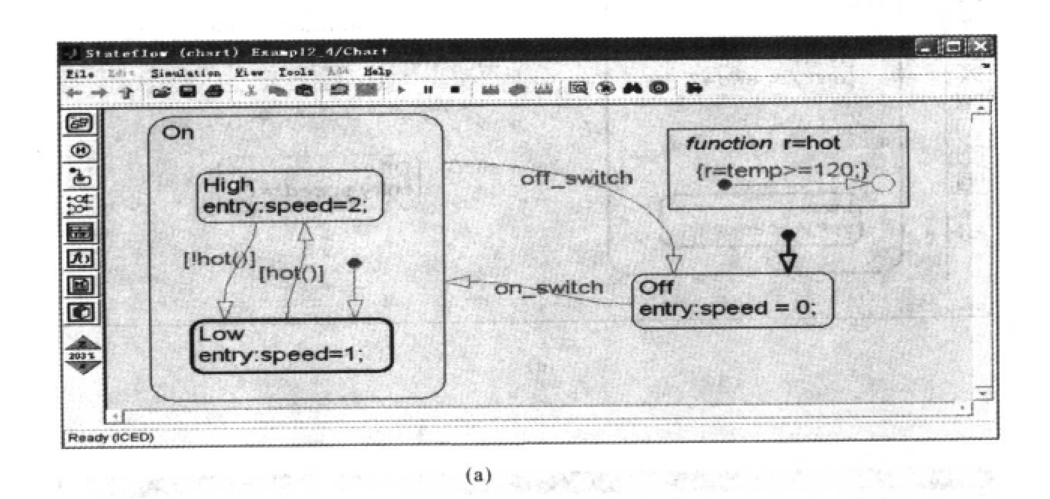
(2) 仿真。将 temp 值置为 130，点击工具栏图标 ▶ 启动仿真。Stateflow 先分析本例的 Stateflow 图是否出现错误，若无错，则给此系统的 Stateflow 图自动生成 S-函数。随后正式进行仿真计算，此时 Simulink 模型中的 Pulse Generater 模块每 0.5 s 向 Stateflow 模块发送一个 temp_event 事件，Stateflow 图在接收到第一个 temp_event 事件后即唤醒处于休眠状态的 Stateflow 图，执行缺省的状态迁移，激活状态 Off，执行状态 Off 的动作，将 speed 置为 0，如图 12.20(a)，(b)所示。

点击 Manual Switch1 模块以产生一个具有上升沿的 on_switch 触发信号，这时 Stateflow 图判断到可以进行状态 Off 到状态 On 的迁移，激活迁移信号线(on_switch 触发的信号线)，停止状态 Off，激活状态 On，如图 12.20(c)，(d)所示。在状态 On 内，要先执行缺省状态迁移，激活其中的 Low 状态，执行 Low 的动作，将 speed 置为 1，如图 12.20(e)，(f)所示。Low 状态与 High 状态之间有个条件迁移关系式，这时 Stateflow 调用图形函数以判断该条件迁移关系式是否成立，对于本例，因为 temp＝130，条件迁移关系式成立，故激活该迁移线，停止状态 Low，激活状态 High，执行状态 High 的动作，将 speed 置为 2，如图 12.20(g)，(h)所示。

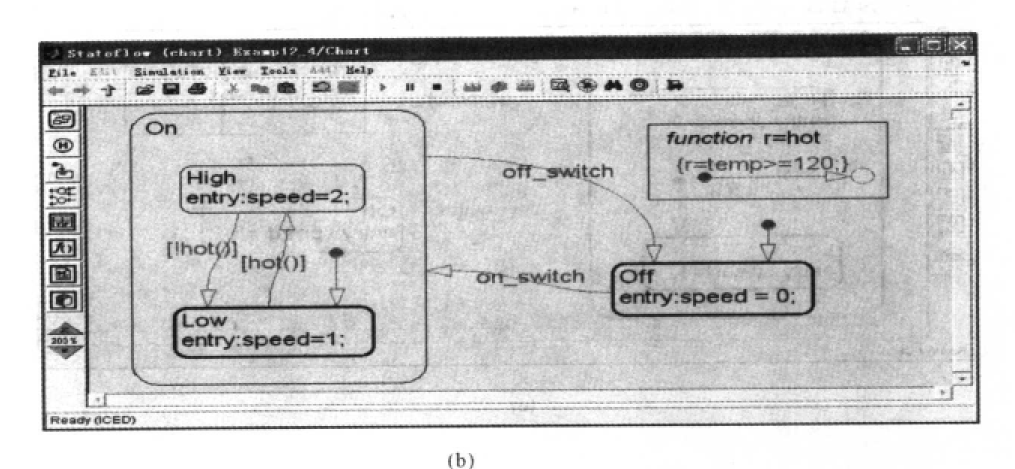
点击 Manual Switch 开关模块，产生一个下降沿的 off_switch 信号，此时 Stateflow 图判断出可以进行状态 On 至状态 Off 的迁移，激活 off_switch 触发的信号线，停止 High 状态，停止 On 状态，激活 Off 状态，执行 Off 状态的动作，将 speed 置为 0，如图 12.20(i)，(j)所示。

注意：例 12.4 基本上实现了例 12.3 完成的功能。但两者之间还是存在小小的差别。在例 12.4 完成状态迁移时，要想激活状态 High，一定需要先激活状态 Low，然后才判断出必须

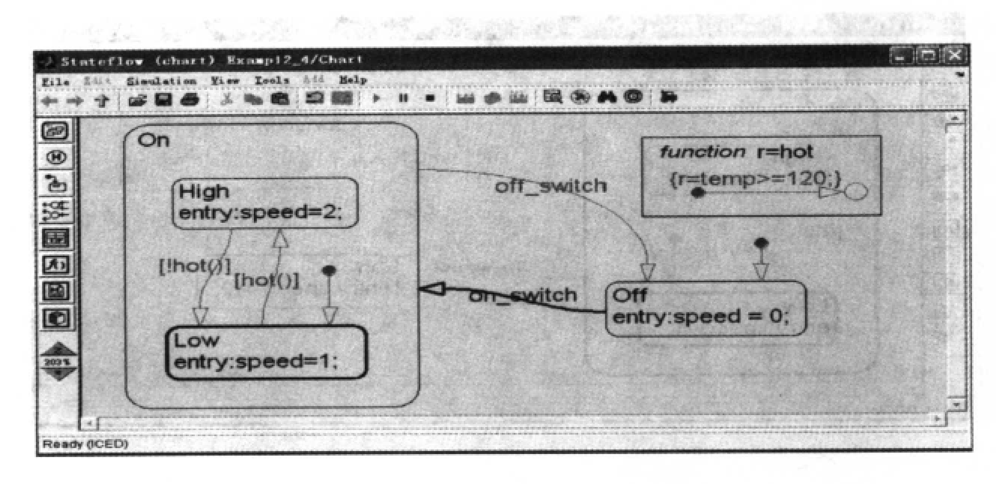
执行状态 Low 到状态 High 的迁移，此时才能激活 High 状态。读者如果仔细观察仿真过程就能看到这样的迁移过程，而且例 12.4 的 speed 值要从 0 变到 2 中间总是要先将它置为 1，然后才置为 2，这是可以从 Scope 模块中观察到的，如图 12.21 所示。



(a)

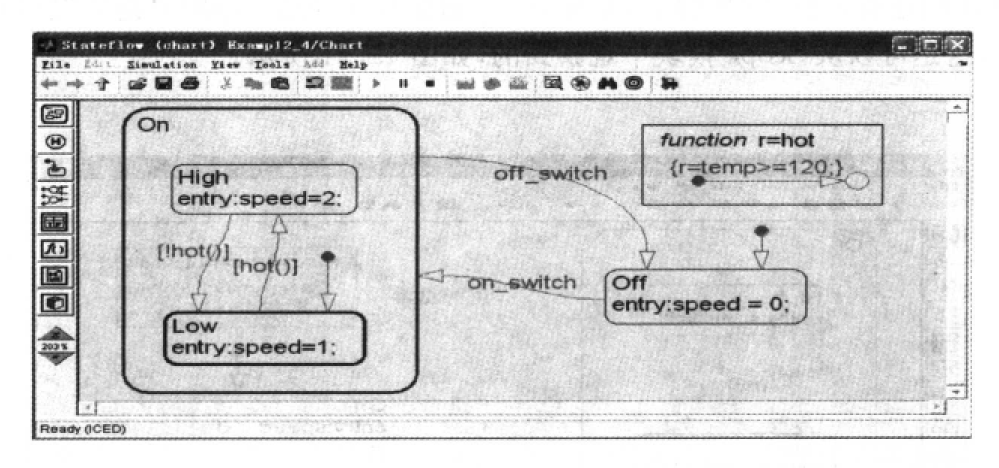


(b)

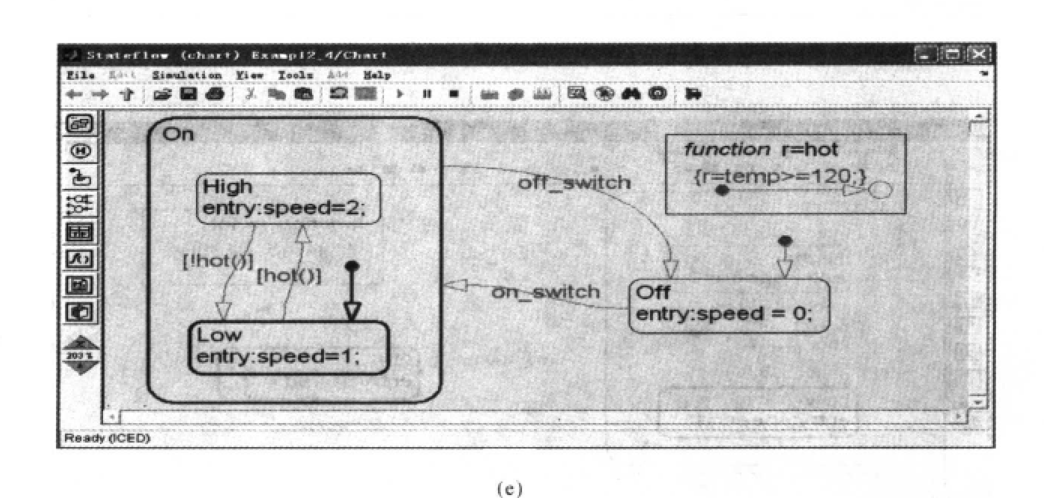


(c)

图 12.20　例 12.4 仿真时的状态迁移示意图



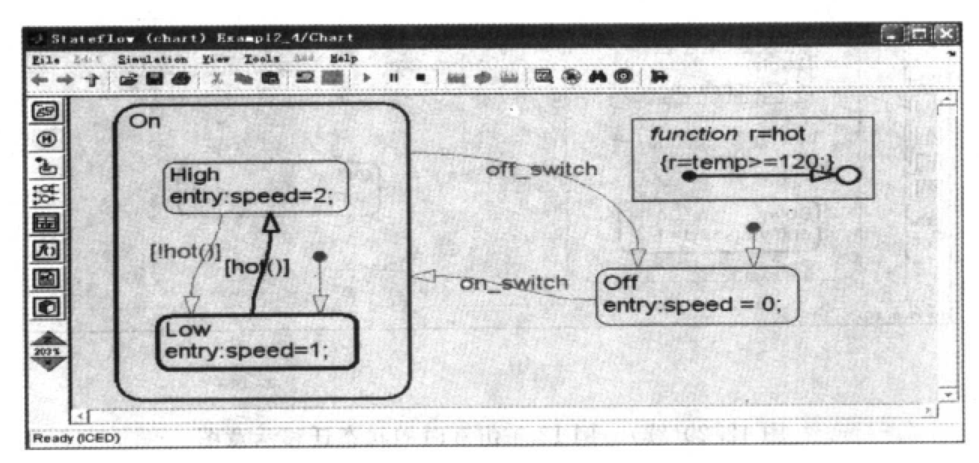
(d)



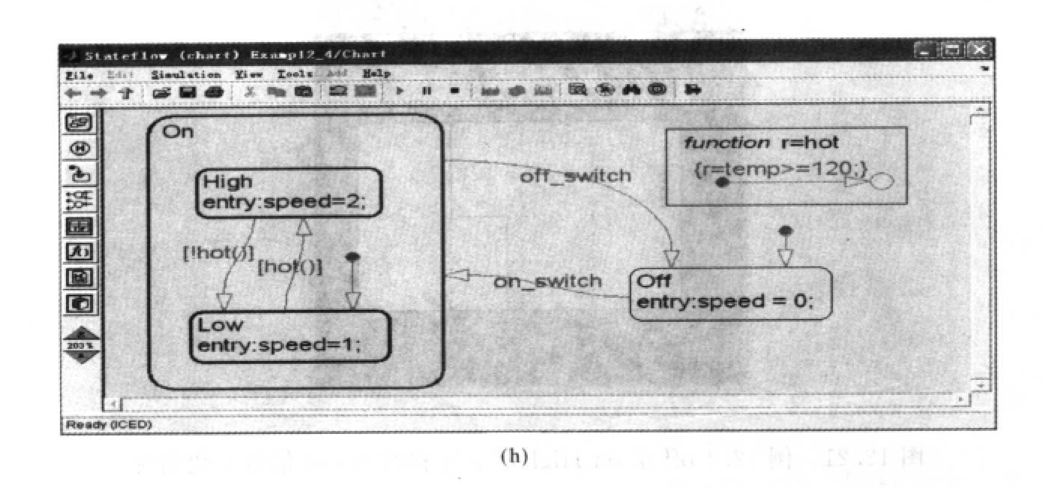
(e)

(f)

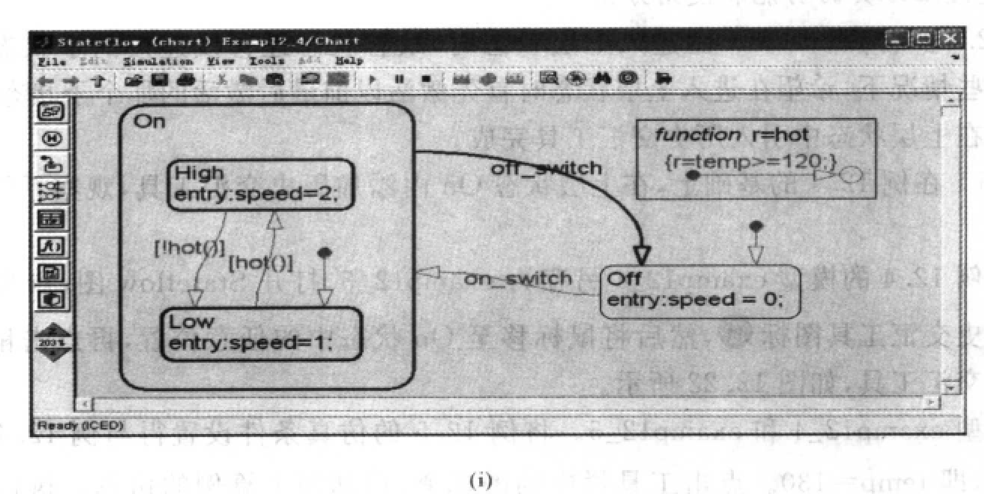
图 12.20(续)　例 12.4 仿真时的状态迁移示意图



(g)

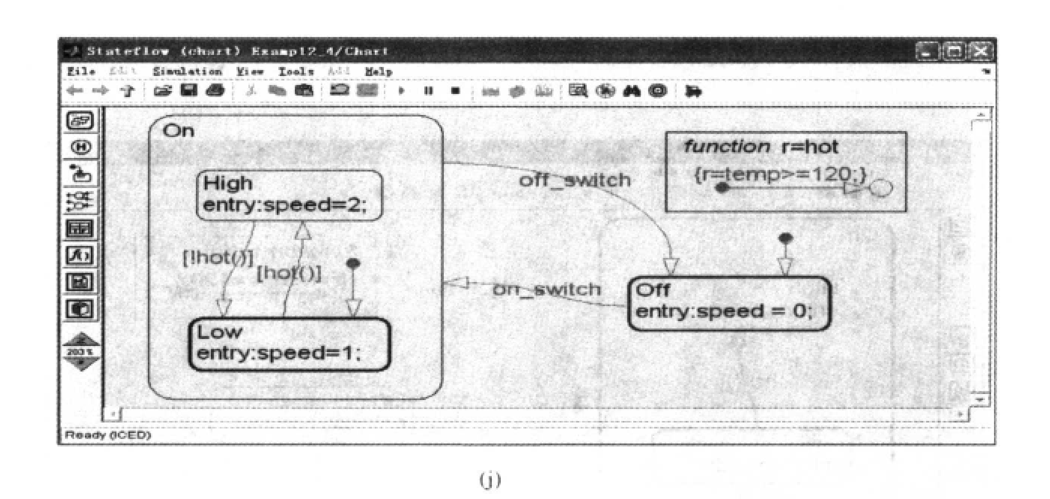


(h)



(i)

图 12.20(续)　例 12.4 仿真时的状态迁移示意图



(j)

图 12.20(续)　例 12.4 仿真时的状态迁移示意图

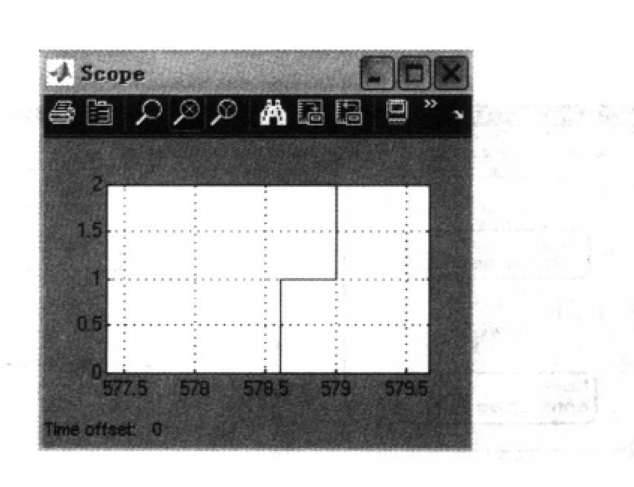


图 12.21　例 12.4 off 至 on High 状态迁移时 Speed 信号变化情况

8. *历史交汇工具的功能和使用方法*

如例 12.4 一样，Stateflow 图进入上层状态时，缺省状态迁移线连接的子状态首先被激活。但在有些情况下，希望在进入上层状态时首先激活以前最后激活的那个子状态，这样的要求可以通过在上层状态中引入历史交汇工具完成。

例 12.5　在例 12.4 的基础上，在上层状态 On 内添加历史交汇工具，观察系统的状态迁移情况。

解　将例 12.4 的模型 exampl2_4 另存为 exampl2_5，打开 Stateflow 图，点击 Stateflow 图左侧的历史交汇工具图标 Ⓗ，然后将鼠标移至 On 状态中的任意位置，再点击鼠标即添加了一个历史交汇工具，如图 12.22 所示。

打开模型 exampl2_4 和 exampl2_5。将例 12.5 的仿真条件设置得与例 12.4 的仿真条件完全相同，即 temp=130。点击工具栏中的图标 ▶，启动两个算例的仿真。执行与例 12.4 完全相同的步骤，直至状态 Off 处于激活状态，如图 12.20(j)所示。在这些步骤中，例 12.4 和例 12.5 的状态迁移是完全相同的。此后按下述步骤执行，观察例 12.5 和例 12.4 状态迁移过程的区别。

改变 Manual Switch1 的开关位置，产生一个上升沿触发信号，on_switch 触发信号有效，

这时 Off 状态停止，激活 On 状态。在例 12.4 中，会激活缺省信号线，随后激活 Low 状态，然后执行图形函数 hot()，判断出 hot()为真，这时激活状态 Low 至状态 High 的迁移线，停止状态 Low，停止两状态之间的迁移线，激活状态 High，系统处于休眠状态，等待下一个触发信号，如图 12.20(e)～(h)所示；而在例 12.5 中，由于有历史交汇工具，Stateflow 图直接激活 High 状态，然后执行图形函数 hot()，判断出确实应该激活 High 状态，就开始等待下一个触发信号。

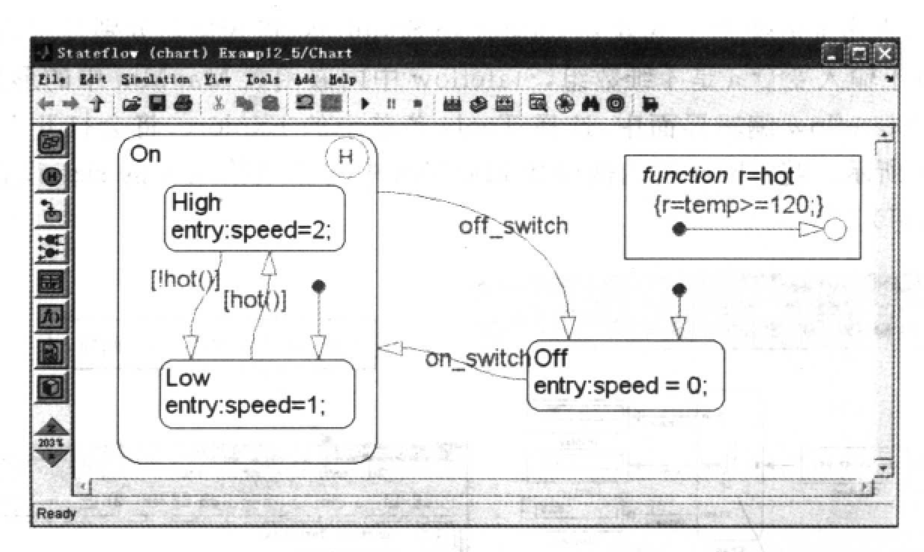


图 12.22　例 12.5 含历史交汇工具的 Stateflow 图

问题：若在改变 Manual Switch1 的开关位置之前，先将 temp 值改为 110，请读者分析例 12.4 和例 12.5 的状态迁移情况。

前面通过几个例子，介绍了 Stateflow 中最基本对象的使用方法，至此用户就可以自行编写简单的 Stateflow 图，对具有相对简单的逻辑判断的系统进行仿真计算了。下面介绍 Stateflow 中经常用到的其他工具的使用方法。

9. 嵌入式 M 函数的设置及其调用

嵌入式 MATLAB 函数使用户可以利用 MATLAB 强大的功能，在 Stateflow 图中编写 MATLAB 语言函数，调用 MATLAB 的各类函数。Simulink 利用嵌入 MATLAB 函数的状态流实现 Simulink 模型中嵌入 MATLAB 模块功能。

要想在 Stateflow 中嵌入 MATLAB 函数，只需在 Stateflow 的编辑器中点击左侧的图标▣，将鼠标移到编辑器中的适当位置，再点击鼠标，并在光标处写入要建立的 MATLAB 内嵌函数名，这时 MATLAB 会自动打开一个编辑 MATLAB 函数的编辑窗口，如图 12.23 所示。内嵌函数的调用类同于图形函数的调用方法，用户可以在状态的动作和迁移过程中对内嵌函数进行反复的调用。

例 12.6　利用 Stateflow 求一组数组的最大、最小及其均值。

解　(1) 建模。首先在 MATLAB 命令窗键入 sfnew，打开一个含 Stateflow 模块的 Simulink 模型窗。打开 Simulink 模型窗中的 Stateflow 模块，打开 Stateflow 编辑器，点击 Stateflow 编辑器左端的内嵌 MATLAB 函数图标▣，将鼠标移到 Stateflow 编辑器中适当的位置，再次点击鼠标，即产生一个内嵌的 MATLAB 函数，在光标处键入内嵌 MATLAB 函数名及其

形参名 MaxMin(x)即可。这时会自动打开一个含 function MaxMin(x)的内嵌 MATLAB 函数编辑窗，在此编辑窗内键入内嵌函数需要完成的程序。例 12.6 的程序如图 12.23 所示。由于本例只是为了求一组数组的最大、最小和均值，这些功能内嵌函数都已实现，因而在 Stateflow 图中，它的调用就非常简单，只需建立一个缺省的状态迁移，在进行状态迁移时执行状态迁移条件动作时调用即可，如图 12.23 所示。例 12.6 只需进行 Stateflow 输入输出数据的设置，在 Stateflow 编辑界面点击 add 菜单下的 data-input from Simulink，将变量名设置为 a；输出变量有 3 个，点击 add 菜单下的 data-output to Simulink，将变量名分别设置为 Xmax，Xmin 和 Xmean。由于输入变量 a 是 4 维数组，Stateflow 中内嵌函数 MaxMin 中的形参 x 也应该是 4 维数组。在 Stateflow 编辑界面中，选择 Tools 菜单中的 Explore，将会打开 Model Explorer，如图 12.24 所示。将 Chart 的内嵌函数 MaxMin 中的函数输入 x 的 size 设置为 4(在 size 列下输入 4 即可)。

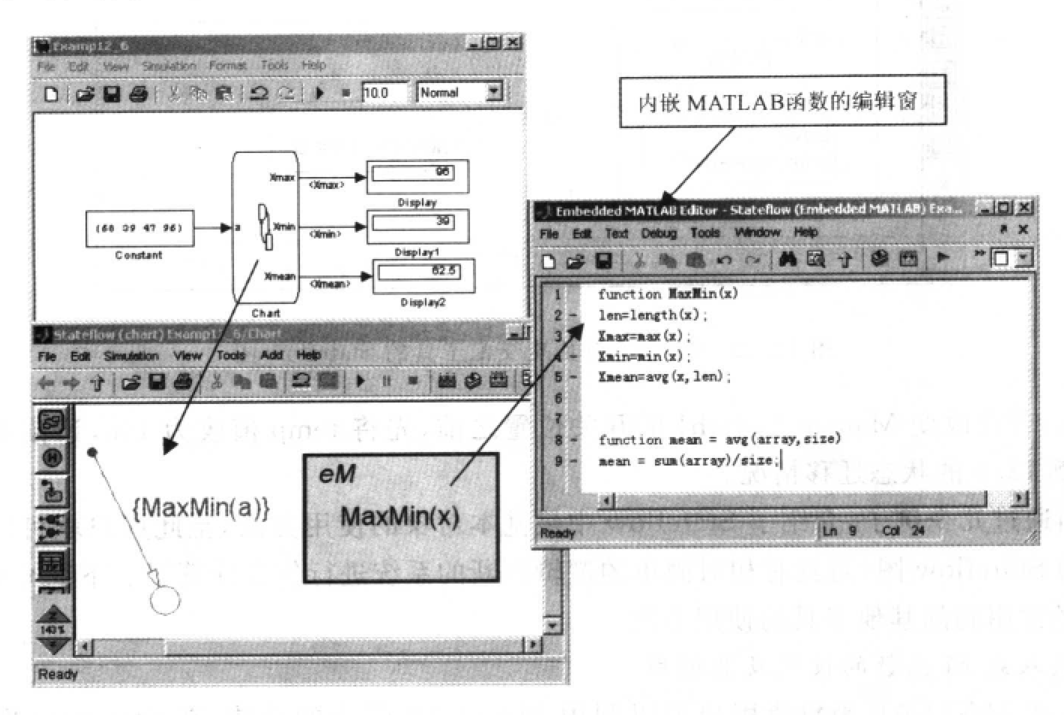


图 12.23　Stateflow 内嵌 M 函数的应用举例

在 Simulink 模型窗口中创建输入模块 Constant，将其输入的常数设置为 4 维数组[68 39 47 96]。添加 3 个显示 Display 模块，并连接对应的信号线，如图 12.23 所示。将此模型存为 Examp12_6。

(2) 仿真。选择 Simulation 菜单下的 Start，启动仿真。仿真开始时，Simulink 先分析 Stateflow 图，如没有错误，就按照 Stateflow 图自动生成 S-函数，并存在当前目录下的 sfprj 子目录下。仿真结束后，3 个 Display 显示模块分别显示出输入数组的最大值、最小值和均值。

上面通过一个很简单的例子说明了 Stateflow 中内嵌 MATLAB 函数的编写、调用及使用方法。需要说明的是这种内嵌 MATLAB 函数不仅可以使用 MATLAB 强大的函数功能，如本例中直接使用 max 和 min 函数，还可以调用用户自行编写的 M 函数，如本例中，函数 MaxMin 还调用了函数 mean。这种情况下，被调用的 M 函数也只需同时写在内嵌 MATLAB 函数编辑器中就行了。

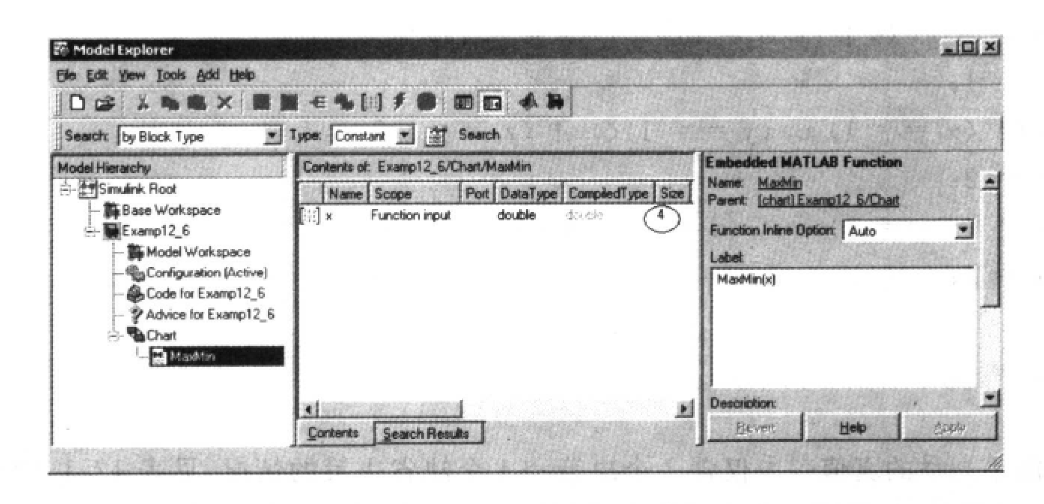


图 12.24　例 12.6 的模型浏览器 Stateflow 图

10. 真值表的设置及其使用

Stateflow 使用函数来处理在 Stateflow 图中须反复处理的动作或判断。在真值表中，用户可以用条件、决策和动作来做逻辑判断，并执行相应的动作。对于纯粹的逻辑来讲，真值表比图形函数更容易编写、维护，也更容易阅读。真值表还可以告诉用户是否对于指定的条件做出了足够的或过多的决策。

Stateflow 使用真值表函数实现逻辑决策及相应动作的执行。Stateflow 真值表含有条件、决策和动作。我们通过算例来说明真值表的设置和工作情况。Stateflow 真值表中条件、决策和动作的排列见表 12.1。

表 12.1　Stateflow 真值表中条件、决策和动作的排列表

Condition(条件)	Decision1(决策 1)	Decision2(决策 2)	Decision3(决策 3)	Default Decision (缺省决策)
x==1	T	F	F	
y==1	F	T	F	
z==1	F	F	T	
Action(动作)	t=1	t=2	t=3	t=4

Condition(条件)列中的每个条件先要判断是真(T)或假(F)，对于上表，就是判断 x==1,y==1,z==1 是否成立。每个条件可以如上表标记为 T,F 或不填(即不论 T 或 F)。每个 Decision(决策)列隐含着各个条件的"与"操作。表 12.1 中 Decision1 列中，当 x==1 为真，而 y==1 和 z==1 同时为假时，Decision1 为真。执行过程中，Stateflow 会从 Decision1 开始判断真值表中的每个决策，如果哪个 Decision 为真，就执行该 Decision 对应的动作。如当 x==1 为真，而 y==1 和 z==1 同时为假时，Decision1 为真，执行动作将 t 置为 1。上表中的最后一个决策称为缺省决策，它包含着除了前面列举的决策外的所有其他决策。如果 Decision1～3 都是假的，则 Default Decision 自动为真，执行其对应的动作，将 t 值置为 4。

用 if-then 语句实现上述的真值表所完成的功能，需编写 if-then 程序如下：

```
if ((x == 1) & ! (y == 1) & ! (z == 1))
    t = 1;
elseif (! (x == 1) & (y == 1)&  ! (z == 1))
    t = 2;
elseif (! (x == 1) & ! (y == 1) &(z == 1))
    t = 3;
else
    t = 4;
end
```

对于简单列举的真值表中仅含 3 个决策和 1 个缺省决策的情况，见表 12.1，用户就需要很多的 elseif 语句，如果再多些决策那么需要的 elseif 语句就会更多。笔者认为采用真值表做逻辑判断是一种非常简单明了的方法。

下面通过一个仿真算例说明真值表的创建方法。

例 12.7 创建如表 12.1 所示的 Stateflow 真值表，通过对 3 个输入变量大小的判断，实现对不同条件下输出变量 t 的相应赋值。

解 (1) 建模。在 MATLAB 命令窗键入 sfnew，打开一个含 Stateflow 模块的 Simulink 模型窗。双击 Simulink 模型窗中的 Stateflow 模块，打开 Stateflow 编辑器，点击 Stateflow 编辑器左端的真值表图标，将鼠标移到 Stateflow 编辑器中适当的位置，再次点击鼠标，即产生一个真值表函数，在光标处键入该真值表函数名及其形参名 t=truthtable(x,y,z)即可。其中 t 是输出形参，x,y,z 是输入形参，truthtable 是本例的真值表函数名。双击已产生的真值表函数图标，打开真值表编辑器，如图 12.25 所示。点击真值表编辑器中的工具可以增添条件编辑表和动作编辑表中的行数。点击工具可以增添 Decision 决策的列数。在真值表编辑器中键入图 12.25 所示的内容，以实现表 12.1 所列举的逻辑判断及动作功能。这样本例的真值表函数就编辑好了。真值表函数的调用类似内嵌 MATLAB 函数和图形函数的调用，在 Stateflow 编辑界面中需调用的地方，键入正确的调用格式。本例中在缺省状态迁移处执行迁移动作时调用，f=truthtable(a,b,c)表示输入实参是 a,b 和 c，输出实参是 f。

这样可知 Stateflow 的输入数据是 a,b 和 c，输出参数是 f，在 Stateflow 界面采用 add 菜单或直接打开其 Model Explorer 添加这些输入输出数据。

在 Simulink 模型窗口中从 Simulink 模型库的 Source 库中添加 3 个 Constant 模块，以便给 Stateflow 输入数据 a,b 和 c。从 Simulink 模型库的 Sinks 库中添加 1 个 Display 模块以便观察仿真结果 f 值。

(2) 仿真。选择 Simulation 菜单下的 Start，启动仿真。仿真开始时，Simulink 先分析 Stateflow 图，如果没有错误，就首先将真值表转换成图形函数，然后按照 Stateflow 图自动生成 S-函数，并存在当前目录下的 sfprj 子目录下。仿真结束后，用户可以右击 Stateflow 编辑器中的真值表函数，从下拉菜单中选择 View Contents 观察由真值表产生的图形函数。

仿真结束后，Display 显示模块显示出 Stateflow 图中的输出 f 值。在本例中，将 a 和 b 置为 0，c 置为 1，仿真结果 f=3。用户可以改变 Simulink 模型窗口中 3 个 Constant 模块的数值，以观察输出结果的变化，并根据真值表的功能，检验显示结果的正确性。

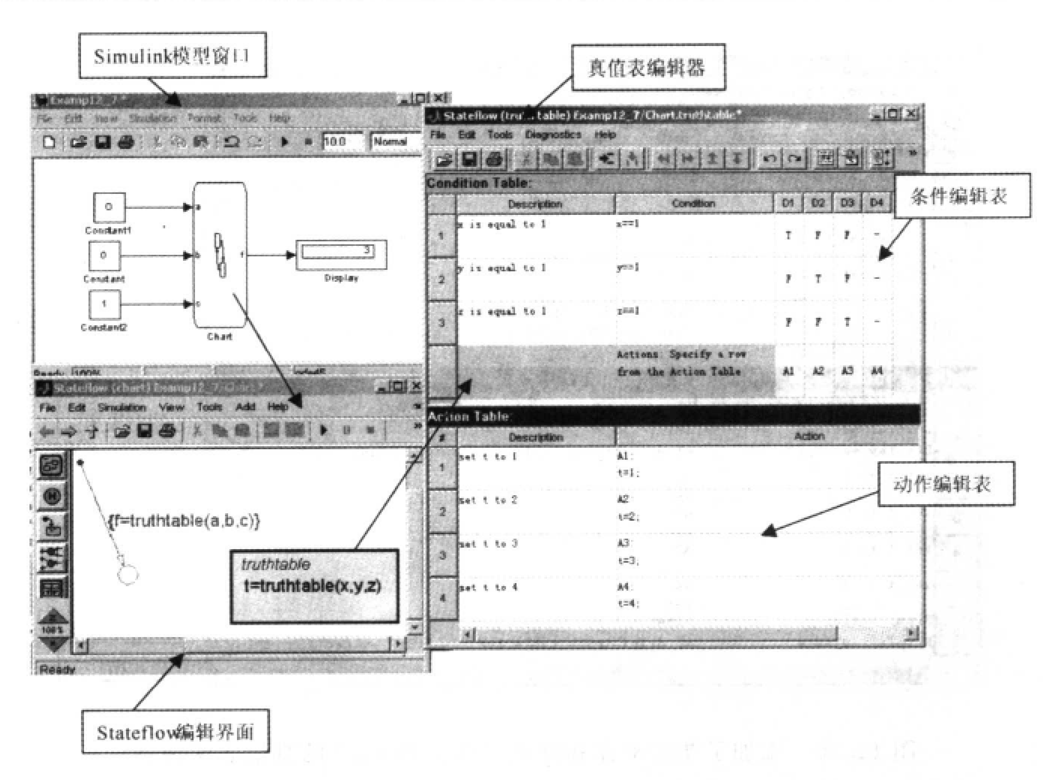


图 12.25　Stateflow 真值表函数举例

除了决策的动作外，Stateflow 还允许用户给真值表函数添加初始和终止动作。初始动作指定任何决策判断前的动作。终止动作指定真值表函数即将退出最后要执行的动作。在动作编辑表中使用标记 INIT 和 FINAL 来为真值表函数指定初始和终止动作。双击真值表函数图标，打开真值表编辑器，右击动作编辑表的第一行，在产生的下拉菜单中选择 Insert Row 即可在动作编辑表最上端产生一空白行。用户可以在此添加初始动作，点击真值表编辑器中的工具可以在动作编辑表最下端插入一个空行，在此空行处添加终止动作。例如，本例中，初始动作和终止动作都添加为显示信息，添加了初始和终止动作的真值表编辑器如图 12.26 所示。仿真时，在每次调用真值表函数时，MATLAB 命令窗口会显示如下提示：

truth table for Examp12.7 entered

truth table for Examp12.7 exited

在此就不多述真值表编辑器工具栏上各图标可以完成的功能。

有时用户在编写真值表时会过多地制订一些决策，这些决策有时是条件编辑表前面的决策可能已经包含的决策，也有的时候用户可能会漏掉一些应用中可能发生的决策。Stateflow 可以在编译时给出用户过多或过少制订决策的错误提示。图 12.27 给出过多决策的示例。在图 12.27 的条件编辑表中，决策 D3(－TT)规定了 FTT 及 TTT，其中 FTT 在 D1 中规定了，而 TTT 也在 D2 中规定了。因而 D3 是属于多余决策。Stateflow 在编译该真值表时，给出了错误提示，如图 12.27 所示。

图 12.28 给出了不足决策的示例。条件编辑表中只规定了 3 种决策，那么可能的决策应该有 8 种，因而 Stateflow 在编辑该真值表时，给出错误信息的同时，指出了其他 5 种可能决策。因而，用户在编写真值表时，可以用像例 12.7 那样，利用真值表的最后缺省决策来表示其他所有前面未规定的决策。

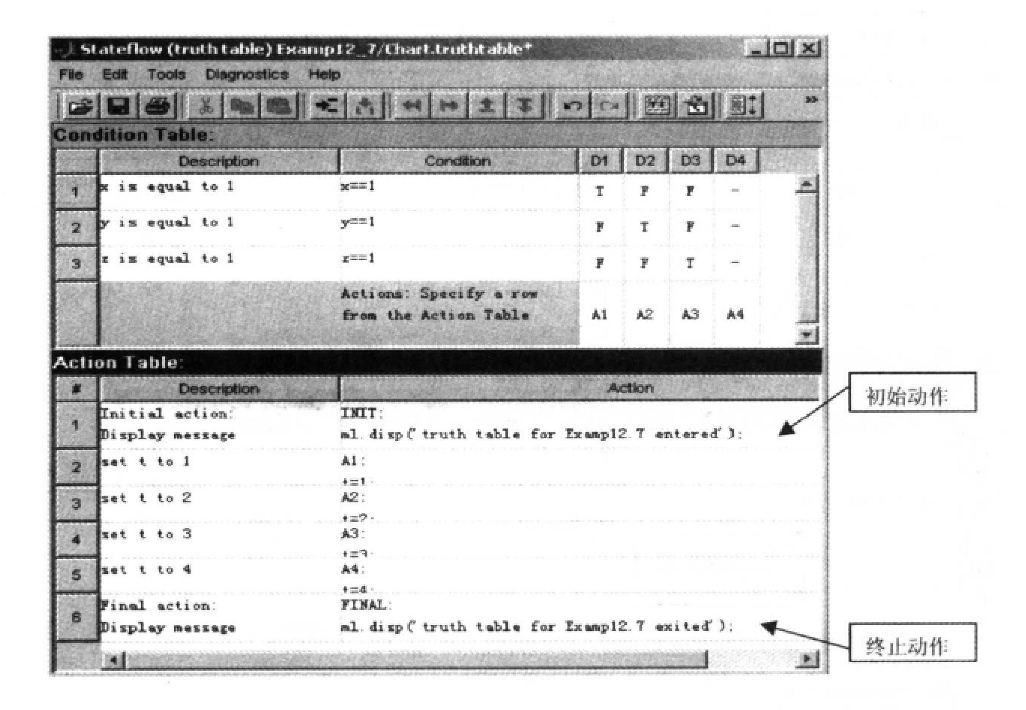


图 12.26　添加了初始动作和终止动作的例 12.7 的真值表编辑器

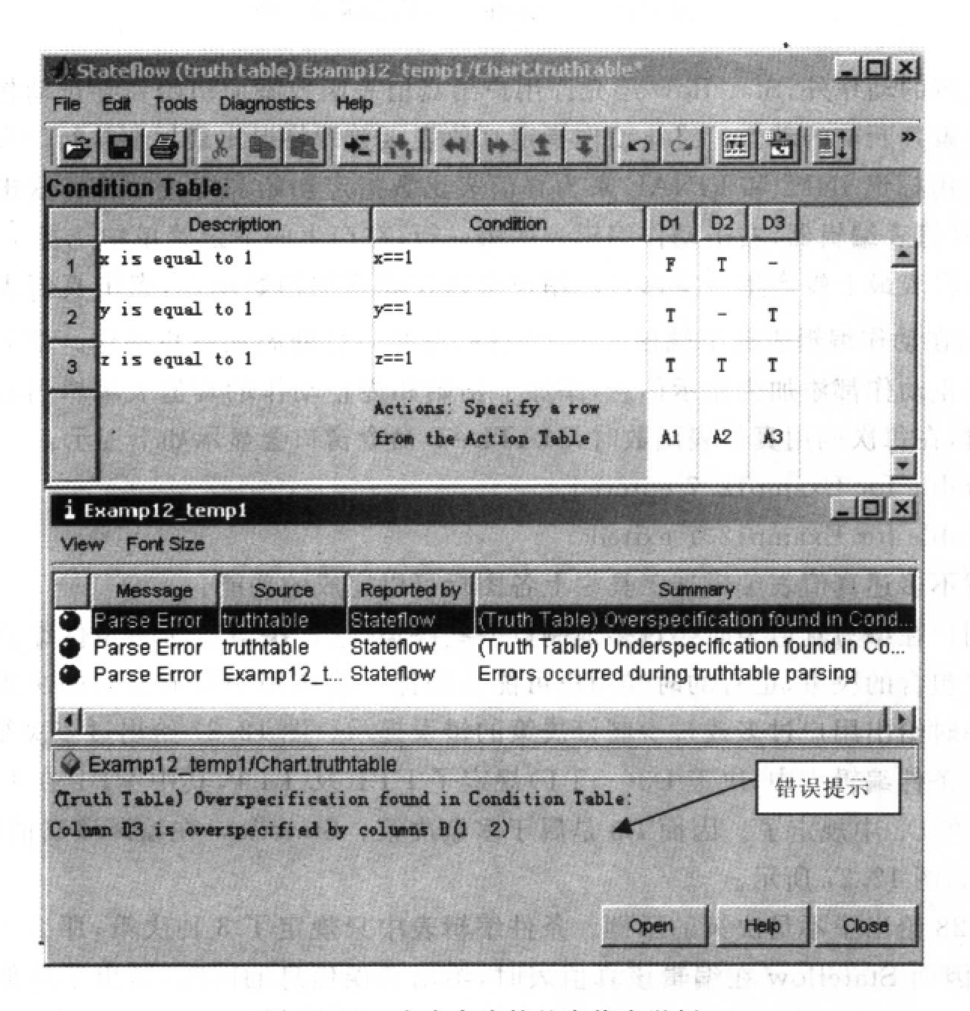


图 12.27　含多余决策的真值表举例

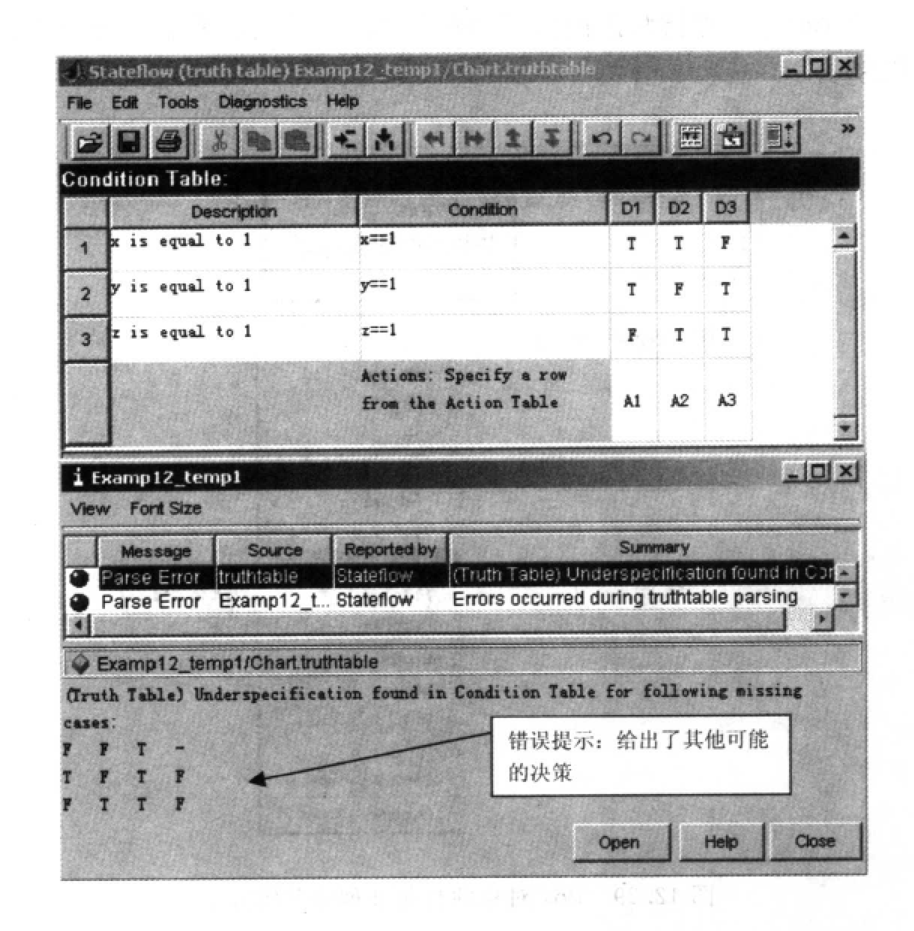


图 12.28　不足决策的真值表举例

11. 用 Box 工具整理状态流图

Boxes 能够很方便地用来整理 Stateflow 图。Boxes 的创建很简单，有几种方法。

首先，利用 Stateflow 编辑界面左边的 Box 工具。点击图标，将鼠标移至 Stateflow 编辑界面的适当位置，再点击鼠标，即可创建一个 Box 对象，在 Box 的问号处写入该 Box 的名称。

其次，可以先建立一个状态，将该状态转换为 Box。右击创建好的状态，在弹出的下拉菜单中选择 Type - Box 即可创建一个 Box 对象。

创建好 Box 工具后，用户可以在此 Box 工具中创建其他的对象以完成一定的逻辑判断功能。

有的时候，用户可能已经建立了一些对象了，这时，可以利用上述的两种方法创建 Box 对象，然后将 Box 框扩大到能够包含所有需要包入的对象。

将所有的对象放入 Box 中后，用户还可以将整个含对象的 Box 打包成一个图形对象，只要右击 Box，在弹出的下拉菜单中选择 Make Contents-Group 或简单地双击该 Box 对象，即可完成打包过程，打包的对象边框变粗；用户也可以右击 Box，在弹出的下拉菜单中选择 Make Contents-Subcharted 隐含 Box 中的对象，将 Box 中的对象变成子图形式，如图 12.29 所示。

对一个 Box 添加数据，可以使 Box 中的所有元素共享该数据。

大多数情况下，Boxes 不改变 Stateflow 图实现的逻辑判断功能，但是在存在并行状态时，它却影响着 Stateflow 图中的激活顺序。Stateflow 图中的 Box 比它右边任何并行状态和 Box

要先激活。在一个 Box 内，并行状态的激活顺序依然是上下、左右的顺序（关于并行状态的问题，下面即将讲述）。

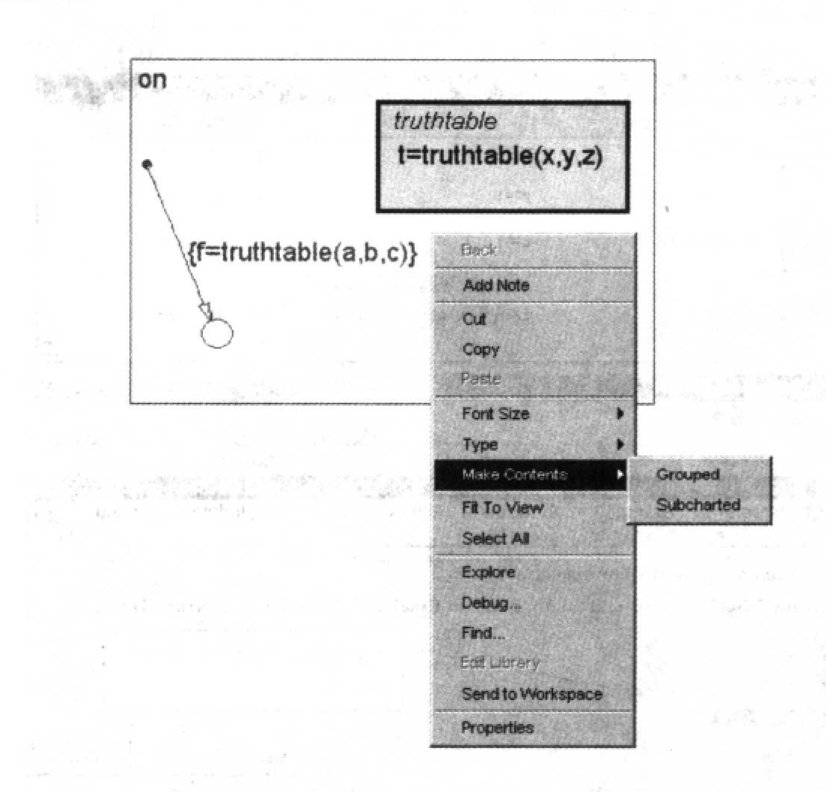


图 12.29 Box 对象的打包和创建分图方法

12. 含并行执行状态的 Stateflow 图

到目前为止，在前面的举例中所创建的状态都是单一状态（Exclusive State 或称为 Or State），单一状态的边界是实线。单一状态的最大特征是同一时间父状态中仅有一个子状态可以处于激活状态。

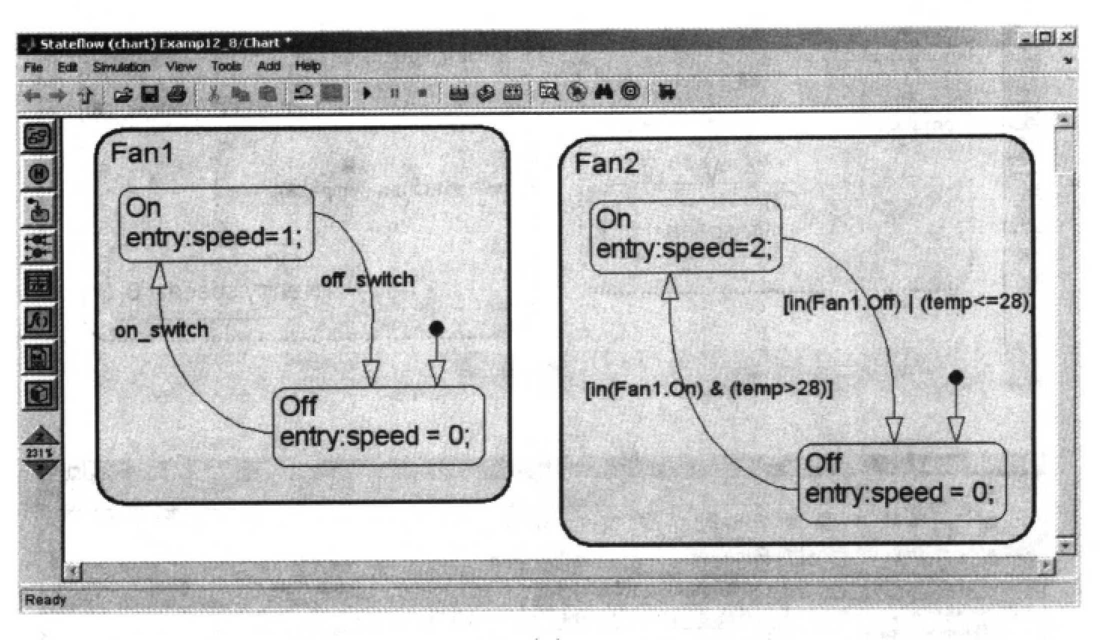
并行状态（Parallel State 或称为 And State）的边界是虚线，并行状态的特点是父状态中的多个子状态可以同时处于激活状态。虽然说并行状态可以同时被激活，但还是存在一个激活顺序的问题。并行状态在状态图中的位置决定着其激活的顺序。位于上部的状态比位于下部的状态较早被激活，同一高度层中，位于左边的状态较位于右边的状态较早被激活。激活的顺序依照上→下，左→右的原则。

下面我们举例来控制两个独立的被控对象，以此说明如何建立并行状态，以及并行状态的激活顺序。

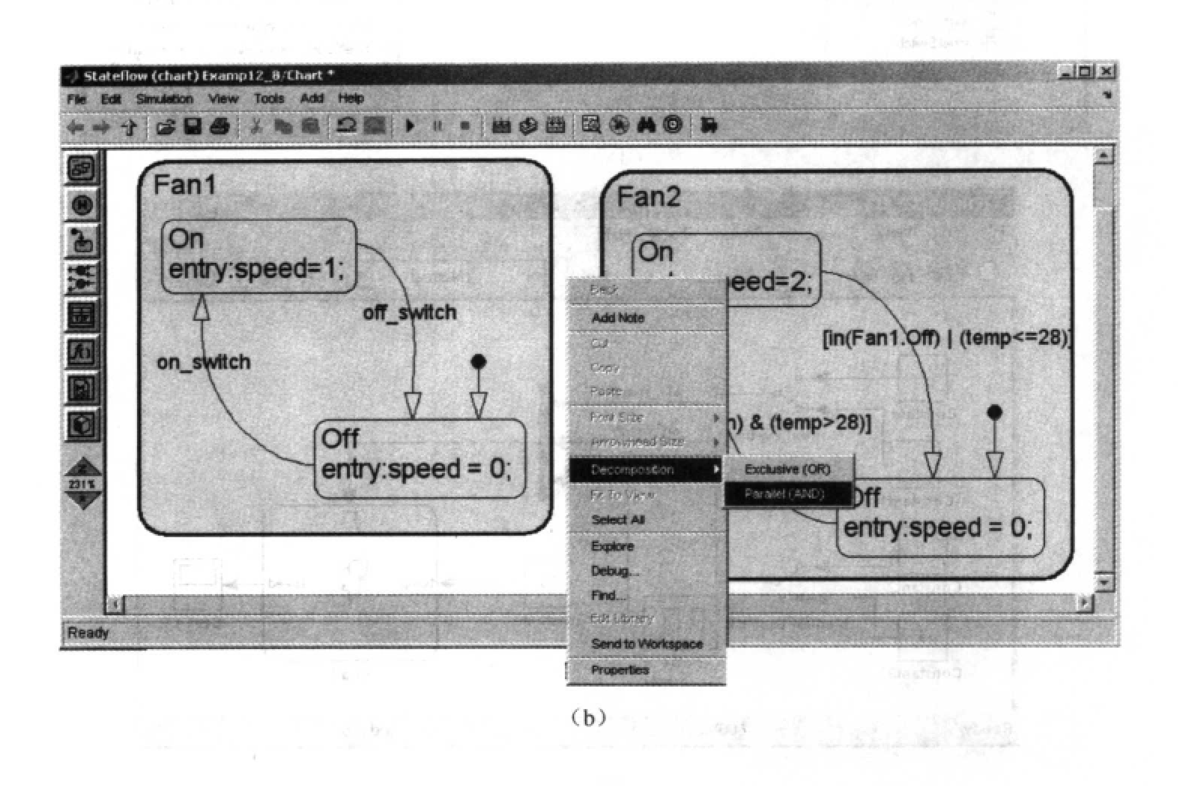
例 12.8 现有一间车间，为了改善工人的工作环境，车间安装了两台排气扇。一般情况下，只需一台排气扇工作，但当室内的温度超过 28℃时，启动第二台排气扇，两台同时工作。这样两台排气扇必须独立控制。我们建立如下模型进行两台排气扇的并行控制。

解 (1) 建模。Stateflow 图的创建。在 MATLAB 命令窗口键入 sfnew，打开一个含有 Stateflow 模块的 Simulink 模型窗口。点击 Stateflow 模块，打开 Stateflow 编辑界面。建立两个含子状态的上层状态 Fan1 和 Fan2，两状态内含的状态、状态迁移及其迁移事件、条件情况如图 12.30(a)所示。状态 Fan2 中的迁移条件[in(Fan1.Off) | (temp<=28)]表示若 Fan2 的子状态 On 是激活状态，当状态 Fan1 中的状态 Off 是激活状态或 temp<=28 为真

时，Fan2 中发生从状态 On 到状态 Off 的迁移；类似地，状态 Fan2 中的迁移条件[in(Fan1. On) & (temp>28)]表示若 Fan2 的子状态 Off 是激活状态，当状态 Fan1 中的状态 On 是激活状态且 temp<=28 为真时，Fan2 中发生从状态 Off 到状态 On 的迁移。

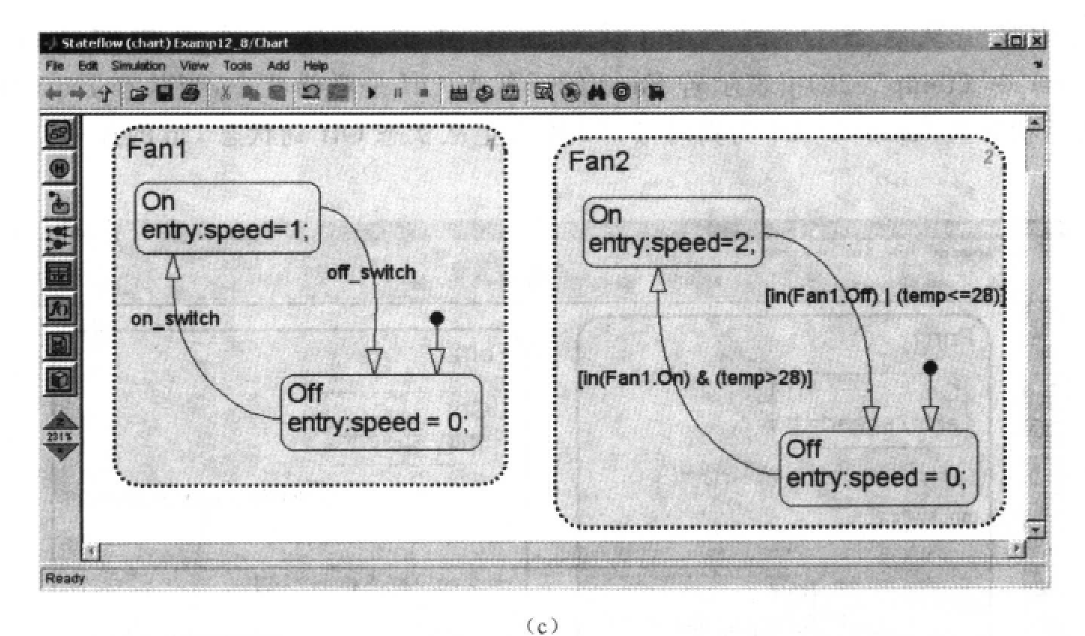


(a)



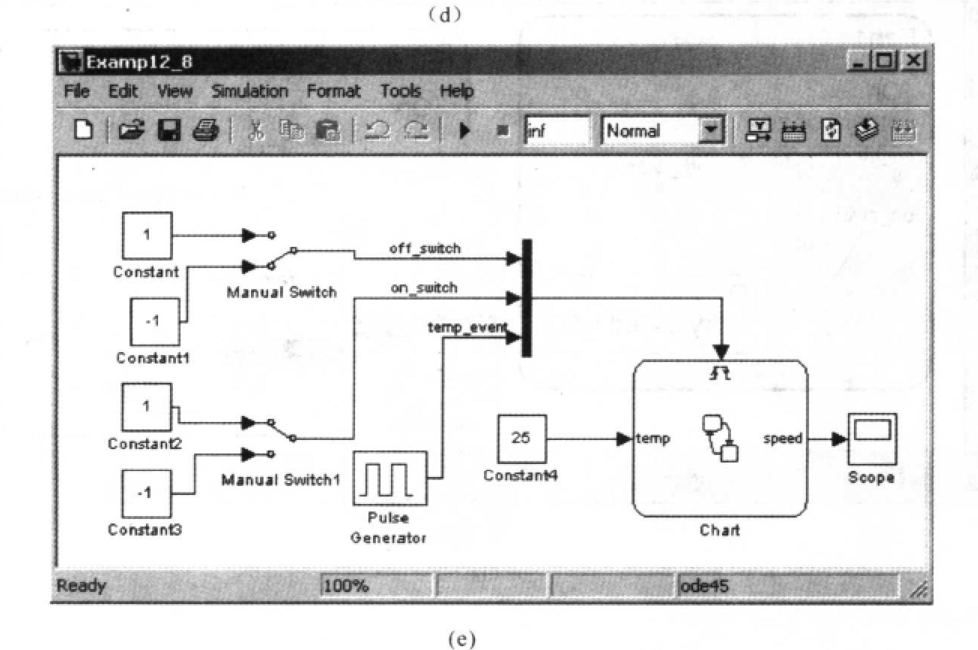
(b)

图 12.30　例 12.8 的建模方法



(c)

(d)



(e)

图 12.30(续)　例 12.8 的建模方法

在 Stateflow 编辑界面的空白处，右击鼠标，在弹出的下拉菜单中选择 Decomposition - Parallel(AND)即可将两个状态 Fan1 和 Fan2 设置为并行状态，表现为该两状态的边框变为虚线，且 Stateflow 按照上下、左右的原则，自动给出此两个并行状态执行的顺序，并在其右上角标出顺序号。(用户可以通过改变状态 Fan1 和 Fan2 的位置来观察 Stateflow 中并行状态执行顺序的变化)，如图 12.30(b)和(c)所示。此 Stateflow 图中的数值 temp 是需要从 Simulink 模型中输入的，除此之外，此 Stateflow 图还需利用 Simulink 模型中的 temp_event 事件以触发此 Stareflow 图每 0.5 s 读一次 temp 数值并判断是否应该状态迁移。Stateflow 图需添加三个 Input from Simulink 的事件，即 off_switch，on_switch 和 temp_event；一个 Input from Simulink 的数值 temp 和一个 Output to Simulink 的数据 speed。将事件 off_switch，on_switch 和 temp_event 的触发沿分别设置成 Falling，Rising 和 Either。添加后，选择 Stateflow 图中的 Tools 菜单下的 Explorer，打开模型管理器 Model Explore，其中的事件和数值设置应如图 12.30(d)所示。

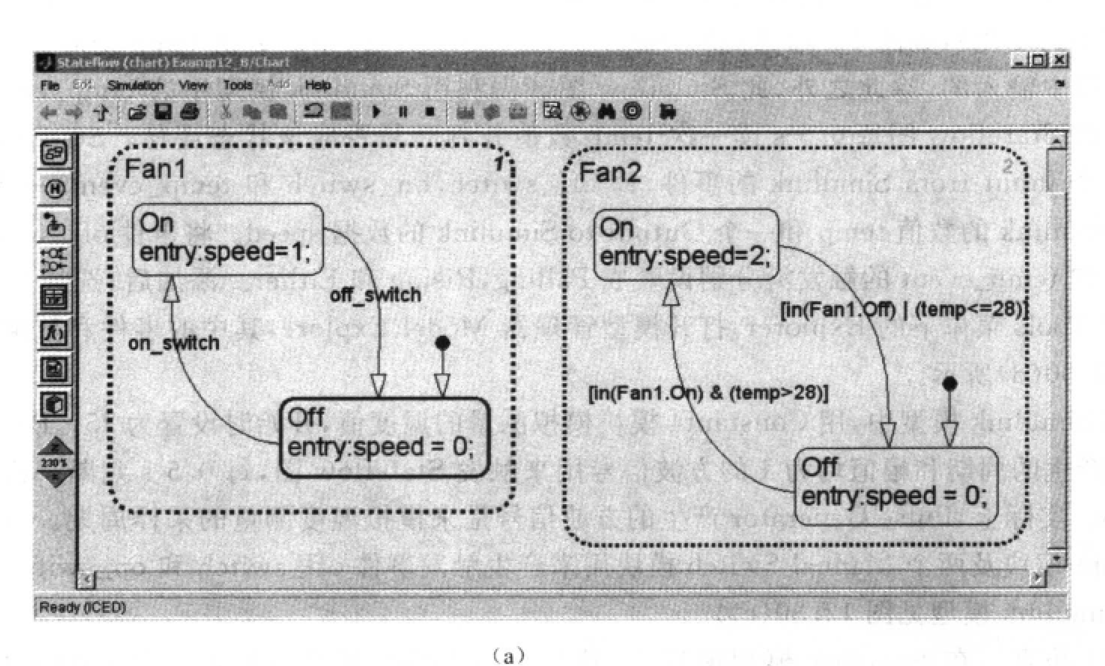
在 Simulink 模型中，用 Constant4 模块模拟测量的温度值，开始时设置为 25。Pulse Generator 产生的周期和幅值均为 1 的方波信号用来触发 Stateflow 图，每 0.5 s 判断一次温度值的大小。实际上，Pulse Generator 产生的方波信号是来模拟温度测量的采样周期。其他 4 个 Constant 模块及两个 Manual Switch 模块用来产生触发事件 off_switch 和 on_switch 的。该例的 Simulink 模型见图 12.30(e)。

(2) 仿真。在 Simulink 模型窗口中，选择 Simulation 菜单下的 Configuration Parameters，将仿真终止时间设置为 *inf*。

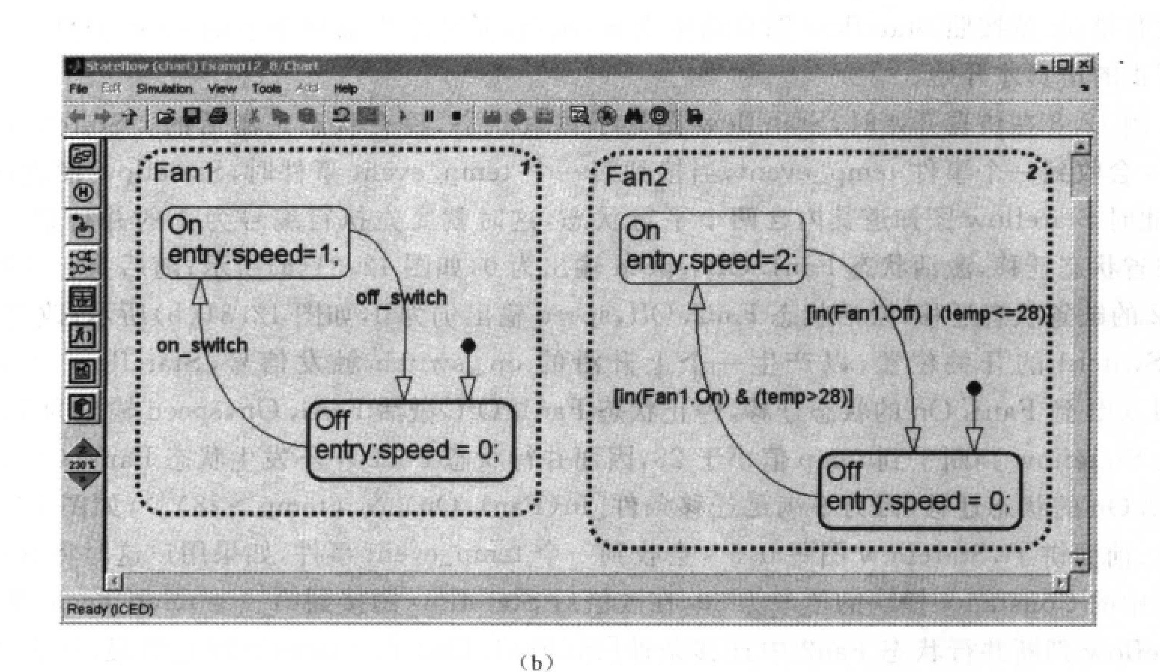
选择 Simulation 菜单下的 Start，启动仿真。仿真开始时，Simulink 先分析 Stateflow 图，如没有错误，就按照 Stateflow 图自动生成 S-函数，并存在当前目录下的 sfprj 子目录下。这时真正的仿真才开始。

例 12.8 在仿真开始时，Stateflow 图处于休眠状态，没有状态是激活的。Stateflow 图每 0.5 s 会收到一个事件 temp_event，当接到第一个 temp_event 事件时，Stateflow 图就被唤醒了，此时 Stateflow 图知道其内含两个平行状态，这时就要先执行编号为 1 的并行状态 Fan1 的缺省状态迁移，激活状态 Fan1. Off，speed 输出为 0，如图 12.31(a)所示；随后执行并行状态 Fan2 的缺省状态迁移，激活状态 Fan2. Off，speed 输出仍为 0，如图 12.31(b)所示；改变 Manual Switch1 的开关位置，以产生一个上升沿的 on_switch 触发信号，Stateflow 产生状态 Fan1. Off 到 Fan1. On 的状态迁移，停止状态 Fan1. Off，激活 Fan1. On，speed 输出为 1，此时，因为 Stateflow 判断了到 temp 值小于 28，因而并行状态 Fan2 中不发生状态 Fan2. Off 到状态 Fan2. On 的状态迁移(因为不满足迁移条件[in(Fan1. On) & (temp>28)])，如图 12.31(c)所示；前面讲了，Stateflow 图每 0.5 s 会收到一个 temp_event 事件，如果用户这时将 Simulink 模型中的 Constant4 模块的值改为 30，在改值后 Stateflow 图接到第一个 temp_event 事件时，Stateflow 判断并行状态 Fan2 中迁移条件[in(Fan1. On) & (temp>28)]满足，即发生状态 Fan2. Off 到状态 Fan2. On 的状态迁移，停止状态 Fan2. Off，激活 Fan2. On，将 speed 值置为 2，如图 12.31(d)所示；如果用户改变 Manual Switch 的开关位置，以产生一个下降沿的 off_switch 触发信号，则再次发生从 Fan1. On 状态至 Fan1. Off 状态的状态迁移，激活 Fan1. Off 状态，将 speed 值置为 0，同时 Fan2. On 状态至 Fan2. Off 状态的迁移条件也满足，也发生 Fan2. On 状态至 Fan2. Off 状态的状态迁移，激活状态 Fan2. Off，将 speed 值置为 0，(见图

12.31(e))。本例中状态迁移的过程及迁移过程中的信号流向详如图 12.31 所示。读者也可以自行调解参数 temp 值的大小及 Manual Switch 和 Manual Switch1 的位置，以观察此例的状态迁移情况。

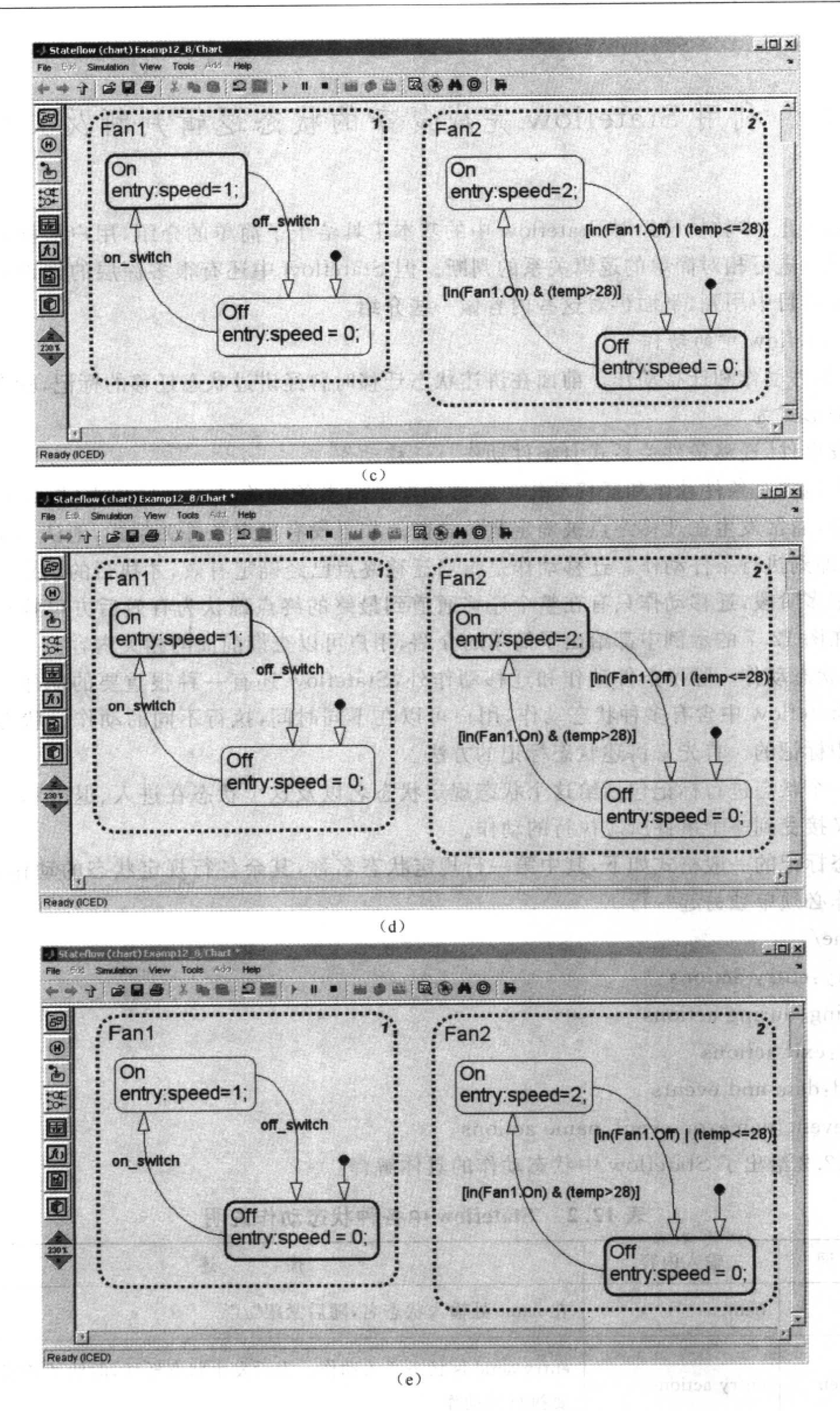


(a)



(b)

图 12.31　例 12.8 系统状态迁移过程



(c)

(d)

(e)

图 12.31(续)　例 12.8 系统状态迁移过程

12.3 利用 Stateflow 完成复杂的状态逻辑判断及其动作

至此为止，本教材已经对 Stateflow 中的基本工具给予了简单的介绍，用户根据前面讲述的知识可以进行相对简单的逻辑关系的判断。但 Stateflow 中还有很多深层的知识可能在以后的实际项目中用到，下面再对这些内容做一些介绍。

1. Stateflow 中的动作

(1)条件动作和迁移动作。前面在讲述状态迁移时曾经讲过状态迁移的标记，状态迁移标记的一般形式为

触发事件[迁移条件关系式]{条件动作}/迁移动作

其中包含有条件动作和迁移动作。条件动作是指当条件关系式一旦成立(即为真)，就执行的动作，通常发生在迁移终点被确定有效之前。如果没有规定条件关系式，则认为条件关系式为真，即刻执行条件动作。迁移动作是指当迁移终点已经确定有效，才执行的动作。如果迁移包含很多阶段，迁移动作只有在整个迁移通道到最终的终点确认为有效后方可执行。这两种动作在图 12.7 的示例中都给出了简单的介绍，用户可以查询前面的相关内容。

(2)状态动作。除了条件动作和迁移动作外，Stateflow 还有一种很重要的动作——状态动作。Stateflow 中含有多种状态动作，用户可以在不同时间，执行不同的动作。状态动作是在状态中标记的。首先来讲述状态标记的方法。

对一个状态进行标记包括给这个状态规定状态名以及这个状态在进入、退出和处于激活状态下又接受到一个事件所应执行的动作。

状态标记的一般格式如下，其中第一行规定状态名称，其余各行规定状态的动作，每个状态的动作必须单独另起一行。

```
name/
entry:entry actions
during:during actions
exit:exit actions
bind:data and events
on event_name:on event_name actions
```

表 12.2 给出了 Stateflow 中状态动作的具体解释。

表 12.2 Stateflow 中各种状态动作说明

关键词	输入内容	描述
无	name	在 name 处输入状态名，随后紧跟“/”
entry 或 en	entry actions	entry actions 状态进入动作。表示发生状态迁移，激活了该状态时需要执行的动作

续 表

关键词	输入内容	描　　述
during 或 du	during actions	during actions 状态仍然激活动作。表示原处于激活的状态受到一个事件的触发，不存在从这个状态发出的状态迁移时，此状态仍处于激活状态需要执行的动作
exit or ex	exit actions	exit actions 状态退出动作。表示存在由此状态发出的有效状态迁移时，该状态退出时执行的动作
bind	data and events	数据事件绑定动作。将数据 data 和事件 events 绑定在此状态上。绑定的数据只能在此状态或其子状态内被改写，其他状态只能读取此数据。绑定的事件由此状态或其子状态广播
on	event_name；on event_name actions	特定事件发生动作。event_name 规定一个特定的事件；on event_name actions 表示当该状态是激活状态且 event_name 规定的事件发生时需要执行的动作

在前面的举例中对状态做了这样的标记

On/

Entry：speed＝1；

表示状态名为 On，当进入状态 On 时，执行状态 entry 进入动作，给变量 speed 赋值。图 12.32 中给出了一个含条件动作、迁移动作和状态动作的 Stateflow 图。

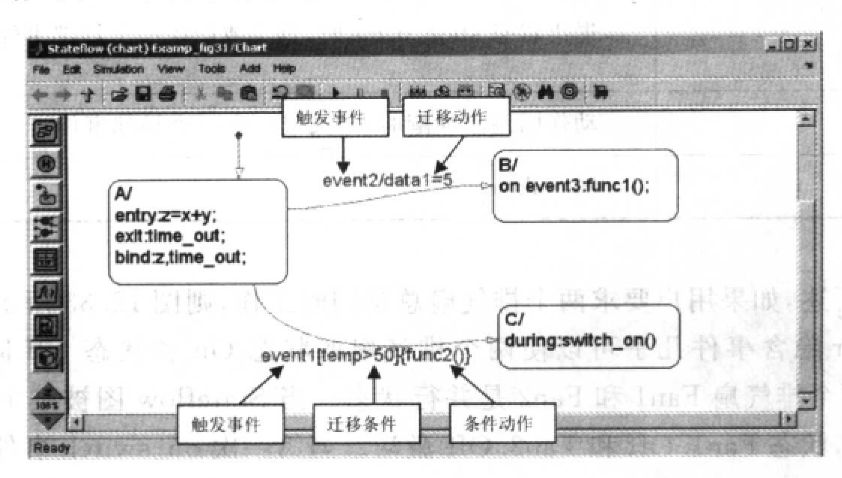


图 12.32　含条件动作、迁移动作和状态动作的 Stateflow 图

当图 12.32 所示的 Stateflow 图被唤醒后，将发生下列迁移或动作：

首先，执行缺省状态迁移，执行状态 A 的 entry 进入动作，即计算 z＝x＋y，事件 time_out 绑定在状态 A 中，激活状态 A。

如果状态 A 是激活状态，Stateflow 图接收到了事件 event2，则状态 A 至 B 的迁移是有效的，执行状态 A 的 exit 退出动作，即广播事件 time_out，停止状态 A，执行状态迁移动作，即将 data1 的值置为 5，激活状态 B。若此时 Stateflow 图接收到了事件 event3，则执行事件 event3

指定的动作，即调用函数 func1()。

如果状态 A 是激活状态，Stateflow 图接收到了事件 event1，判断迁移条件 temp > 50 是否为真，如果为假，则停止。如果为真，执行条件动作，即调用函数 func2()，状态 A 至 C 的迁移是有效的，执行状态 A 的 exit 退出动作，即广播事件 time_out，停止状态 A，激活状态 C。在状态 C 处于激活状态时，如果此时 Stateflow 图接收到了事件 event3，则不会发生任何状态迁移，状态 C 仍处于激活状态，执行 C 状态的 during 动作，即调用函数 switch_on()。

2. Stateflow 中的隐含事件

在 12.3 节中，针对 Stateflow 的事件，介绍了事件添加、事件的触发、事件属性的设置等。事实上除了上节介绍的真实的事件外，Stateflow 中还含有一些隐含的事件。

当 Stateflow 图被唤醒时或进入到某个状态或从某个状态退出或某个内部数据(非输入数据)赋值时，Stateflow 会定义并触发某种事件。这些事件是 Stateflow 自动定义触发的，非用户定义、添加的，故称为隐含事件。

表 12.3 给出了 Stateflow 中的隐含事件的类型和含义。

表 12.3 Stateflow 中的隐含事件类型及含义

隐含事件	含义
change(data_name) chg(data_name)	当变量 data_name 的数值发生变化时，定义或产生一个局部事件
enter(state_name) en(state_name)	进入状态 state_name 时，定义或产生一个局部事件
exit(state_name) ex(state_name)	退出状态 state_name 时，定义或产生一个局部事件
wakeup	动作图刚刚唤醒时，定义或产生一个局部事件
tick	同 wakeup

如例 12.8 所述，如果用户要求两个排气扇总是同时工作，则图 12.33 所示的 Stateflow 图中利用两个 enter 隐含事件几乎可以使两个排气扇的状态 On 和状态 Off 同时进入。在图 12.33 所示中，两个排气扇 Fan1 和 Fan2 是并行状态。当 Stateflow 图被某个事件唤醒时，执行缺省状态迁移，状态 Fan1.Off 和 Fan2.Off 激活。当第一次 on_switch 事件发生时，发生从状态 Fan1.Off 到 Fan1.On 的状态迁移。当 Fan1.On 的进入动作执行时，即刻广播一个隐含的局部事件 en(Fan.On) == 1。这个隐含事件触发从状态 Fan2.Off 到 Fan2.On 的状态迁移。类似地，当系统发生从状态 Fan1.On 到 Fan1.Off 的状态迁移时，广播隐含局部事件 en(Fan1.Off)，这时触发从 Fan2.On 到 Fan2.Off 的状态迁移。

3. 动作中的瞬时逻辑操作(Temporal Logic Operators)

这部分介绍如何在动作中使用瞬时逻辑操作。瞬时逻辑操作是布尔操作，对状态流内部事件反复进行计数。图 12.34 所示是一个含有瞬时逻辑操作的示例。

在瞬时逻辑操作中有几个一般的概念需要先介绍一下。

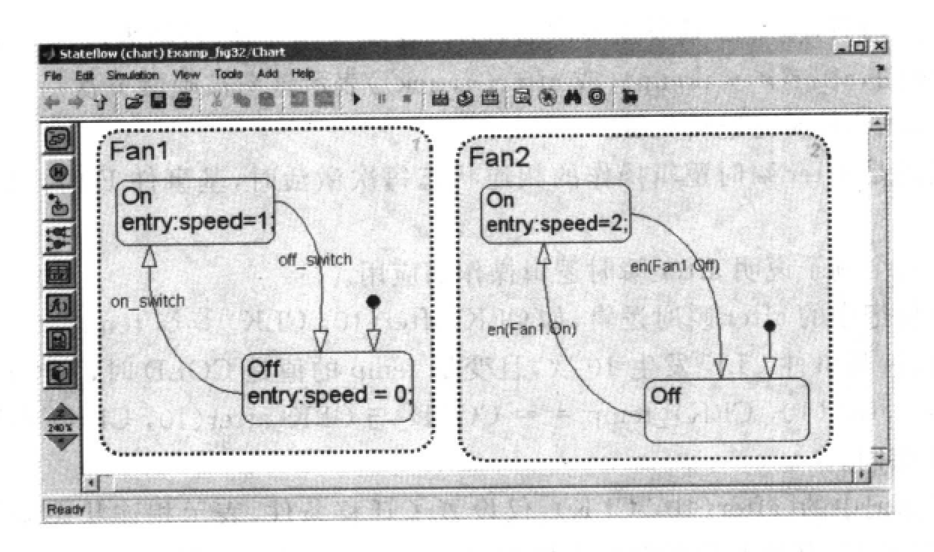


图 12.33　隐含事件应用举例

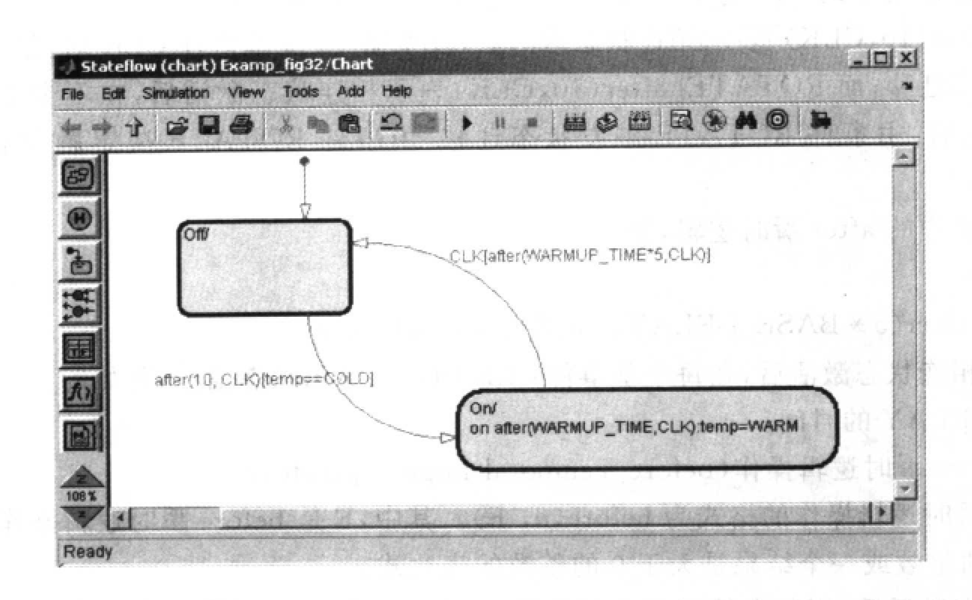


图 12.34　含瞬时逻辑操作的 Stateflow 图示例

瞬时操作中被反复计数的 Stateflow 的内部事件称做基事件(base event)。任何的 Stateflow 事件(包括 enter 事件 、exit 事件或 data change 事件等隐含事件)都可以做为瞬时操作的基事件。

对于没有 Simulink 输入事件的 Stateflow 图,用户可以使用隐含事件 wakeup(或 tick)来唤醒 Stateflow 图。瞬时逻辑操作只能发生在从状态发出的状态迁移条件中或发生在状态动作中。这就意味着瞬时逻辑操作不能用在缺省迁移或流程图的迁移中。

含瞬时操作的迁移出发的状态或其动作中出现瞬时操作的状态称为瞬时操作相连状态(temporal operator′s associated state)。

常用的瞬时逻辑操作有以下几种:

(1) after 瞬时逻辑操作(after Temporal Logic Operator)。

after 瞬时逻辑操作的格式为 after(n, E),其中,E 是 after 瞬时逻辑操作的基事件,n 为大于 0 的整数或一个结果是整数且大于 0 的表达式。

相连状态激活后，基事件 E 发生 n 次后，after 逻辑操作为真，否则为假。对于没有输入事件的 Stateflow 图，after(n，wakeup)（或 after(n，tick)）表示该图被唤醒 n 次后，after(n，wakeup)（或 after(n，tick)）瞬时逻辑为真。

需要指出的是 after 瞬时逻辑操作的相连状态每次激活时，基事件 E 的计数器都被复位为 0。

下面给出几个例子说明 after 瞬时逻辑操作的应用。

状态迁移标记中的 after 瞬时逻辑，如 CLK[after(10, CLK) && temp == COLD]表示相连状态激活后，基事件 CLK 发生 10 次，且变量 temp 的值是 COLD 时，发生从相连状态出发的状态迁移。after(10, CLK)[temp == COLD]与 CLK[after(10, CLK) && temp == COLD]的意义相同。

状态迁移标记中的[after(10, CLK)]仅设置了迁移条件，表示相连状态激活后，发生了 10 次基事件 CLK 后，任何其他的触发事件均可触发从相连状态出发的状态迁移。

状态迁移标记中的 CLK[after(10,CLK)] 和 ROTATE[after(10,CLK)表示不同的含义。CLK[after(10,CLK)]表示相连状态激活后，发生第 10 次基事件 CLK 时，触发从相连状态出发的状态迁移；而 ROTATE[after(10,CLK)表示相连状态激活后，在不少于 10 次基事件 CLK 发生后，由事件 ROTATE 触发状态迁移，由事件 ROTATE 决定触发状态迁移的时间。

状态动作中的 after 瞬时逻辑，如

```
on/
on after(5 * BASE_DELAY, CLK): status('on');
```

表示在相连状态激活后，在每个基事件 CLK 周期内显示状态信息，开始时间是 CLK 发生 5×BASE_DELAY 的时间。

(2) before 瞬时逻辑操作(before Temporal Logic Operator)

before 瞬时逻辑操作的格式为 before(n, E)。其中，E 是 before 瞬时逻辑操作的基事件，n 为大于 0 的整数或一个结果是大于 0 的整数的表达式。

相连状态激活后，当基事件 E 发生的次数小于 n 时，before 逻辑操作为真，否则为假。对于没有输入事件的 Stateflow 图，before(n，wakeup)（或 before(n，tick)）表示当该图被唤醒次数少于 n 时，before(n，wakeup)（或 before(n，tick)）瞬时逻辑为真。

需要指出的是 before 瞬时逻辑操作的相连事件每次激活时，基事件 E 的计数器都被复位为 0。

下面给出几个例子说明 before 瞬时逻辑操作的应用。

状态迁移标记中的 before 瞬时逻辑，如 ROTATION[before(10, CLK)]表示相连状态激活后，基事件 CLK 发生的次数少于 10 次时，由事件 ROTATION 触发状态迁移，由事件 ROTATION 决定触发状态迁移的时间。

状态动作中的 before 瞬时逻辑，如

```
on/
on before(MAX_ON_TIME, CLK): temp++;
```

表示在相连状态激活后，在每个基事件 CLK 周期中，将 temp 值加 1，直到 CLK 事件发生 MAX_ON_TIME 次为止。

(3) at 瞬时逻辑操作(at Temporal Logic Operator)。

at 瞬时逻辑操作的格式为 at(*n*, E),其中,E 是 at 瞬时逻辑操作的基事件,*n* 为大于 0 的整数或一个结果是大于 0 的整数的表达式。

相连状态激活后,当第 *n* 次基事件 E 发生时,*at* 逻辑操作为真,否则为假。对于没有输入事件的 Stateflow 图,at(*n*,wakeup)(或 at(*n*,tick))表示当该图被第 *n* 次唤醒时,at(*n*,wakeup)(或 at(*n*,tick))瞬时逻辑为真。

需要指出的是 at 瞬时逻辑操作的相连事件每次激活时,基事件 E 的计数器都被复位为 0。

下面给出几个例子说明 at 瞬时逻辑操作的应用。

状态迁移标记中的 at 瞬时逻辑,如 ROTATION[at(10, CLK)]表示相连状态激活后,在基事件 CLK 发生第 10 次时,由事件 ROTATION 触发状态迁移;而 at(10, CLK)表示相连状态激活后,第 10 次发生的基事件 CLK 触发状态迁移。

状态动作中的 at 瞬时逻辑,如

```
on/
on at(10, CLK): status('on');
```

表示在相连状态激活后,当基事件 CLK 第 10 次发生时,显示状态信息。

(4) every 瞬时逻辑操作(every Temporal Logic Operator)。

every 瞬时逻辑操作的格式为 every(*n*, E),其中,E 是 every 瞬时逻辑操作的基事件,*n* 为大于 0 的整数或一个结果是大于 0 的整数的表达式。

相连状态激活后,基事件 E 每发生 *n* 次时,every 逻辑操作为真,否则为假。对于没有输入事件的 Stateflow 图,every(*n*,wakeup)(或 every(*n*,tick))表示当该图每第 *n* 次被唤醒时,every(*n*,wakeup)(或 every(*n*,tick))瞬时逻辑为真。

需要指出的是 every 瞬时逻辑操作的相连事件每次激活时,基事件 E 的计数器都被复位为 0。因而这种操作通常是在状态动作内使用。如

```
on/
on every(10, CLK): status('on');
```

表示在相连状态激活后,基事件 CLK 每发生 10 次,显示一次状态信息。

瞬时逻辑操作实际上是产生一个隐含事件,这种方法并不在 Stateflow 模型中创建任何新的事件,可以理解为基事件(base events)积累多时而产生的一个隐含事件。比如,如果用户希望在状态 A 激活后的 10 个时钟周期时产生从状态 A 发出的状态迁移。用前面掌握的方法就是添加一个新的事件(如 ALARM),当状态 A 激活后的 10 个时钟周期时由 ALARM 事件触发从状态 A 出发的状态迁移。另一种比较简单的方法就是利用我们这里介绍的瞬时逻辑操作法。在系统时钟事件 CLK 的基础上建立触发从状态 A 发出的状态迁移的瞬时逻辑操作 CLK[after(10,CLK)]或 after(10,CLK)。

4. Stateflow 中的事件广播

在 Stateflow 的动作中广播一个事件是非常有用的同步并行状态的方法。事件的反复广播亦可产生周期的过程。

因为 Stateflow 是单线程工作的,因而当它接收到一个事件时,它必须中断当前的活动来处理被广播事件产生的动作。用户可以通过状态动作或迁移动作来广播事件。

在进行复杂的 Stateflow 过程分析时，一定要清楚下列几点正常 Stateflow 过程的基本公理：

(1) 当一个状态是激活的时，它的上层（或父）状态一定也是激活的；

(2) 单一（Exclusive）分解的状态或图，不得有超过一个的激活子状态；

(3) 并行（Parallel）分解的状态或图中，如果某状态是激活状态，那么其处于高优先级的并行状态（Stateflow 图中的位置决定优先级）必定也是激活的状态。

下面通过举例来说明 Stateflow 是如何广播事件的。

(1)通过迁移动作进行直接事件广播。

直接事件广播的格式是

　　　send(事件名，状态名)

利用有效事件名直接进行事件广播的格式是

　　　状态名．事件名

图 12.35 给出了一个直接事件广播的示例。在图 12.35 中，并行状态 A 的子状态 A1 到 A2 的状态迁移上，有一个迁移动作 send(E1,B)，该迁移动作完成将事件 E1 发送到并行状态 B 中。send(E1,B)命令等同于使用有效事件名 B. E1 进行事件广播。

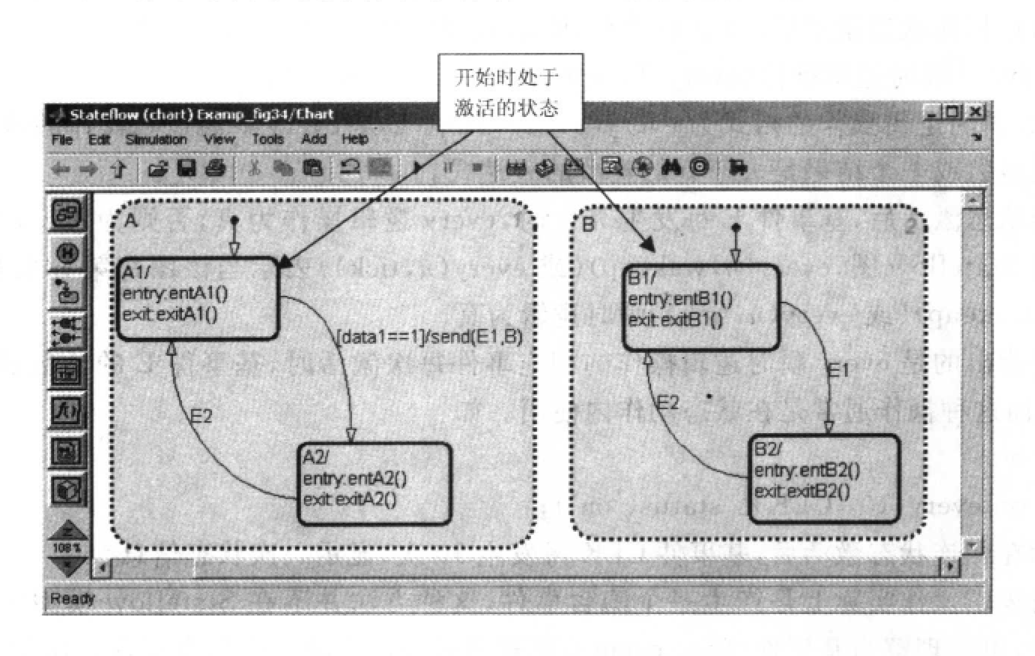


图 12.35　通过状态迁移动作进行事件的直接广播

在图 12.35 所示中，Stateflow 图开始处于休眠状态，并行子状态 A. A1 和 B. B1 是激活的。根据公理，并行上层状态 A 和 B 都是激活的。这时假设一个事件发生，唤醒了 Stateflow 图。如果条件[data1＝＝1]是真。Stateflow 执行次事件的过程为：

Stateflow 图的根部检查此事件是否能产生有效的状态迁移，结果是没有；

这时状态 A 开始检查是否存在该状态内的状态迁移，因为条件[data1＝＝1]是真的，存在状态 A. A1 到 A. A2 的状态迁移；

状态 A. A1 执行 exit 退出动作(exitA1())，停止状态 A. A1；执行状态迁移动作 send(E1，B)，即将事件 E1 广播给状态 B；

以后开始执行广播事件 E1 引发的动作，因为 B 是激活的，接收到事件 E1 后，状态 B 检查

到事件 E1 可以引发 B. B1 到 B. B2 的状态迁移；

状态 B. B1 执行 exit 退出动作(exitB1())，停止状态 B. B1；

激活状态 B. B2，状态 B. B2 执行 entry 进入动作(entB2())，至此广播的事件 E1 引发的过程结束，Stateflow 继续事件广播前的过程；

激活状态 A. A2，执行状态 A. A2 的 entry 进入动作(entA2())；

Stateflow 图再次休眠，等待下一个触发事件唤醒。

(2)通过条件动作进行事件广播。图 12. 36 给出一个通过条件动作进行事件广播的示例。开始时，图 12. 36(a)所示 Stateflow 图处于休眠状态，并行子状态 A. A1 和 B. B1 处于激活状态。事件 E1 发生，唤醒 Stateflow 图。事件 E1 发生后，Stateflow 图的行动过程如下：

Stateflow 图的根部是两个并行状态 A 和 B，依据从上到下、从左到右的优先级原则，先判断状态 A。因为 A 的子状态 A. A1 是激活的，那么 A 一定也是激活的，这时执行状态 A 的 during 动作(durA())。

状态 A 检查是否有由 E1 能触发的状态迁移，检查结果是存在这样有效的状态迁移，从状态 A. A1 到 A. A2，同时还存在条件动作。

条件动作是在条件一旦满足时就执行的，一般早于状态迁移的发生。此例先执行条件动作，广播事件 E2；此时状态 A. A1 还是处于激活状态。

由于广播的事件 E2，Stateflow 要先中断事件 E1 引发的过程，先执行事件 E2 引发的过程。

事件 E2 再次激活 Stateflow 图，先判断状态 A，状态 A 执行 during 动作(durA())。

状态 A 检查不存在由事件 E2 引发的有效状态迁移。

判断状态 B，状态 B 执行 during 动作(durB())；状态 B 检查到一个由 E2 触发的有效状态迁移，从状态 B. B1 到 B. B2，状态 B. B1 执行 exit 动作(exitB1())，停止状态 B. B1，激活状态 B. B2，执行状态 B. B2 的 entry 动作(entryB2())。

至此，由事件 E2 触发的 Stateflow 行为执行完毕，此时 Stateflow 图中状态 A. A1 和 B. B2 是处于激活的状态，如图 12. 36(b)所示。下面继续事件 E1 触发的行为。

状态 A. A1 执行 exit 退出动作(exitA1())；

停止状态 A. A1；

激活状态 A. A2；

执行 A. A2 的 entry 进入动作(entA2())；

判断状态 B，状态 B 执行 during 动作(durB())；

状态 B 判断其内部不存在由事件 E1 引发的状态迁移，状态 B. B2 执行 during 动作(durB2())；

Stateflow 图再次处于休眠状态，等待下一个事件来唤醒。

本例经过事件 E1 触发，通过条件动作广播事件 E2，Stateflow 图中最后激活的状态是 A. A2 和 B. B2，如图 12. 36(c)所示。

(3)通过状态动作进行事件广播。图 12. 37 给出了一个通过状态动作进行事件广播的示例。图 12. 37(a)所示的 Stateflow 图开始时处于休眠状态，并行子状态 A. A1 和 B. B1 处于激活状态。事件 E1 发生，唤醒 Stateflow 图。事件 E1 发生后，Stateflow 图的行动过程如下：

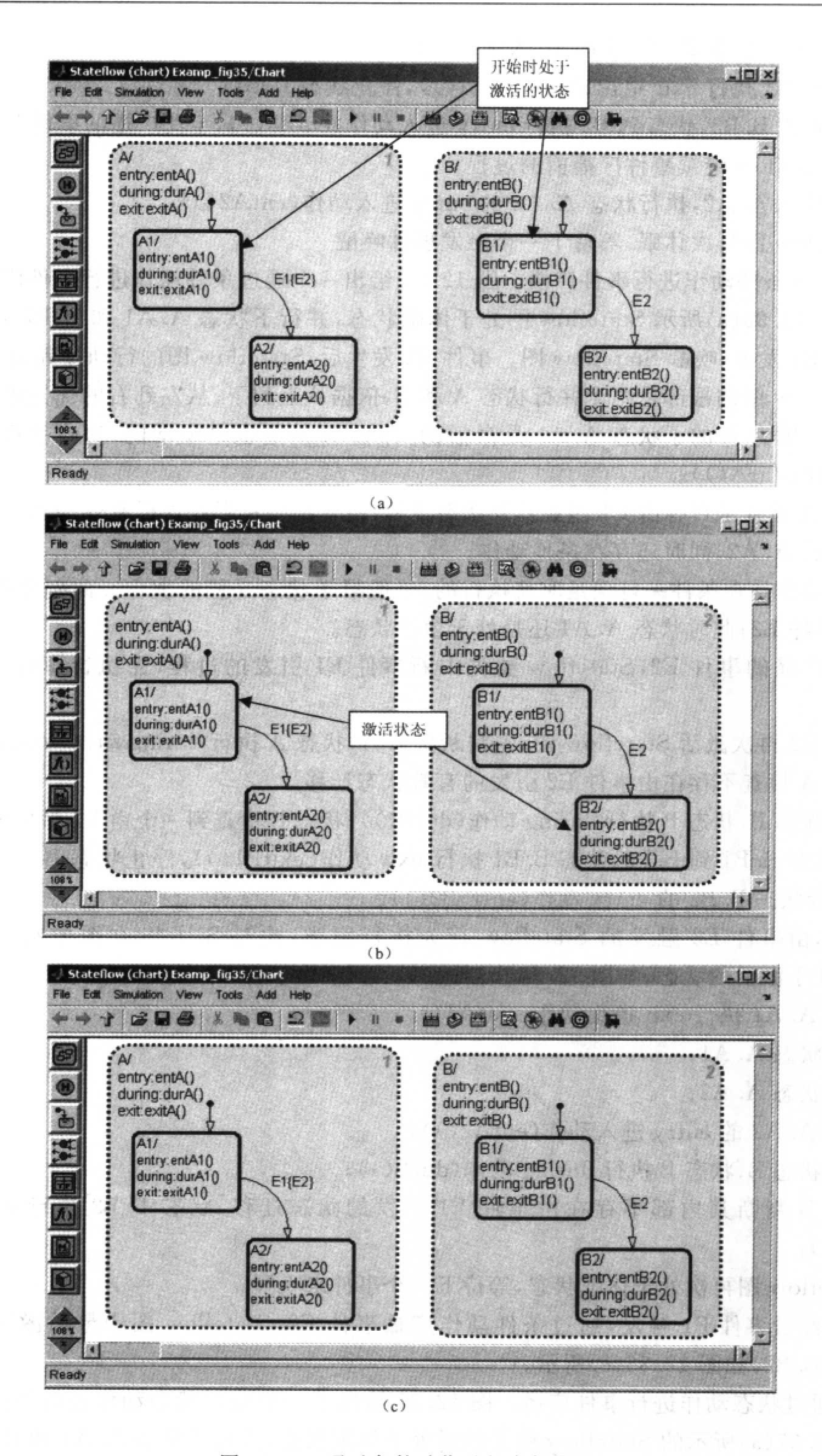


图 12.36 通过条件动作进行事件广播

Stateflow 图的根部是两个并行状态 A 和 B，依据从上到下、从左到右的优先级原则，先判断状态 A。因为 A 的子状态 A. A1 是激活的，那么 A 一定也是激活的，这时执行状态 A 的 during 动作(durA())；

执行状态 A 的 on E1 动作，广播事件 E2；(during 和 on 事件动作的执行顺序由他们在状态中标记中的位置决定，先写的先执行)

由于广播了事件 E2，Stateflow 要先中断事件 E1 引发的过程，先执行事件 E2 引发的过程；

事件 E2 再次唤醒 Stateflow 图，先判断状态 A，状态 A 执行 during 动作(durA())；

状态 A 检查不存在由事件 E2 引发的有效状态迁移；

判断状态 B，状态 B 执行 during 动作(durB())；

状态 B 检查到一个由 E2 触发的有效状态迁移，从状态 B. B1 到 B. B2，状态 B. B1 执行 exit 动作(exitB1())；

停止状态 B. B1；

激活状态 B. B2；

执行状态 B. B2 的 entry 进入动作(entryB2())；

至此，由事件 E2 触发的行为执行完毕，此时 Stateflow 图中状态 A. A1 和 B. B2 是处于激活的状态，如图 12. 37(b)所示。下面继续事件 E1 触发的行为。

状态 A 检查是否有由 E1 能触发的状态迁移，检查结果是存在这样有效的状态迁移，从状态 A. A1 到 A. A2，状态 A. A1 执行 exit 动作(exitA1())；

停止状态 A. A1；

激活状态 A. A2；

执行 A. A2 的 entry 进入动作(entA2())；

判断状态 B，状态 B 执行 during 动作(durB())；

状态 B 判断其内部不存在由事件 E1 引发的状态迁移，状态 B. B2 执行 during 动作(durB2())；

Stateflow 图再次处于休眠状态，等待下一个事件来唤醒。

本例由事件 E1 的触发，通过状态动作广播事件 E2，Stateflow 图中最后激活的状态是 A. A2 和 B. B2，如图 12. 37(c)所示。

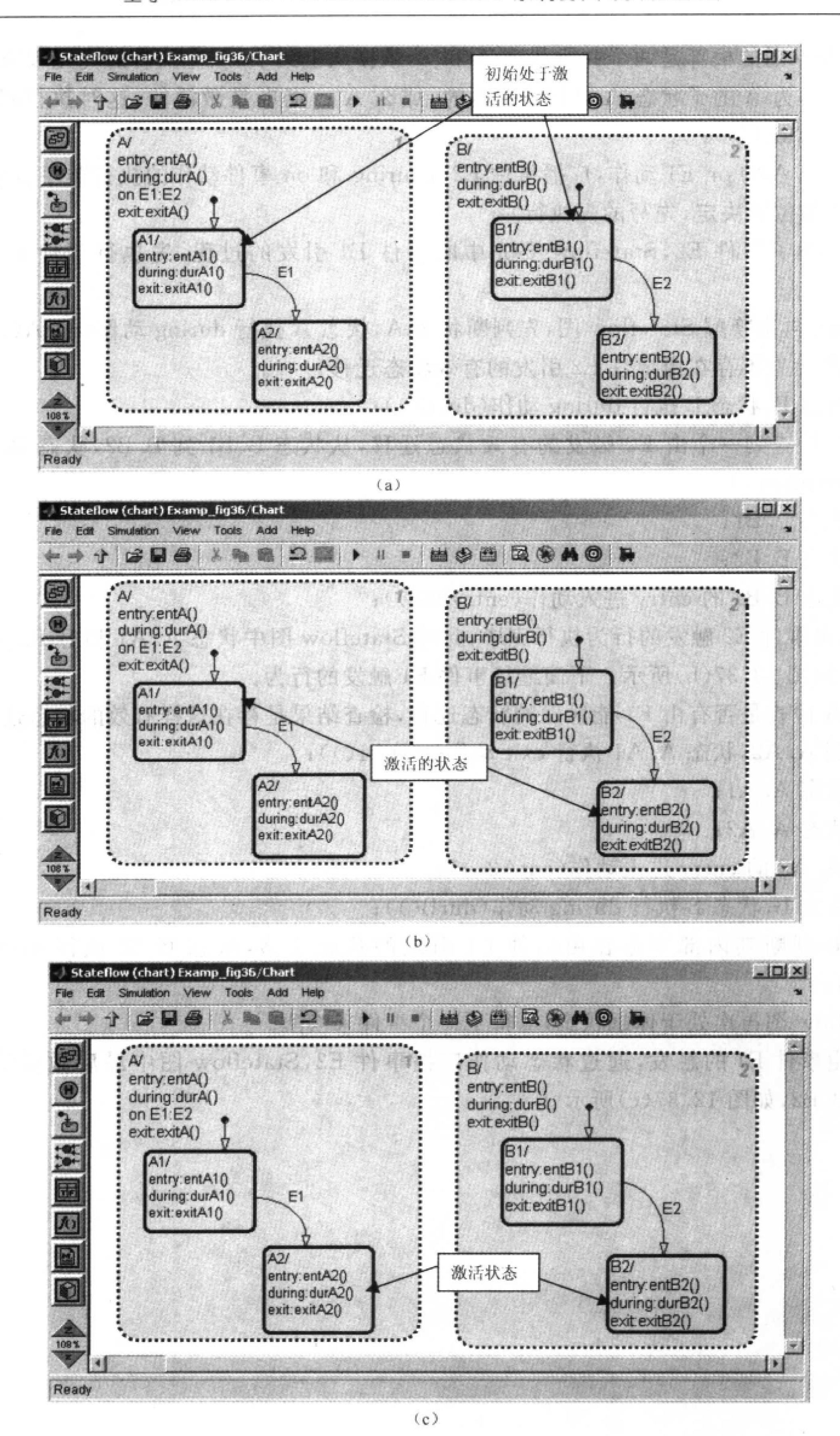


图 12.37　通过状态动作进行事件广播

习　题

12.1　现有一间车间，为了改善工人的工作环境，安装了 3 台排气扇。要求：

（1）3 台排气扇同时通电；

（2）一般情况下，只需一台排气扇工作，但当室内的温度超过 28℃时，启动第二台排气扇，两台同时工作。

（3）当车间从事特种加工（作为一种事件输入）时，打开第三台排气扇，由 3 台排气扇同时工作。

试建立 Stateflow 模型模拟该车间 3 台排气扇的工作情况。

12.2　在 MATLAB 命令窗口键入 fuelsys，打开一个对燃油系统进行容错控制的系统。系统中模拟了各种传感器的故障模式下，容错系统是如何进行工作的，并可计算相应故障模式下系统输出的结果。感兴趣的读者可以仔细分析该系统。

第十三章　反馈控制系统的数学模型及设计工具

反馈系统的数学模型在系统分析和设计中起着很重要的作用，基于系统的数学模型，就可以用比较系统的方法对之进行分析，同时，一些系统的方法也是基于数学模型的，这就使得控制系统的模型问题显得十分重要。

13.1　数学模型的表示方法

线性时不变(LTI)系统模型包括传递函数模型(tf)，零极点增益模型(zpk)，状态空间模型(ss)和频率响应数据模型(frd)。

13.1.1　传递函数模型

线性系统的传递函数模型可以表示成复数变量 s 的有理函数式为

$$G(s)=\frac{b_m s^m+b_{m-1}s^{m-1}+\cdots+b_1 s+b_0}{s^n+a_{n-1}s^{n-1}+a_{n-2}s^{n-2}+\cdots+a_1 s+a_0}$$

调用格式：G = tf (num, den)

其中，num = [b_m　b_{m-1}　…　b_1　b_0]，den = [1　a_{n-1}　a_{n-2}　…　a_1　a_0] 分别是传递函数分子和分母多项式的系数向量，按照 s 的降幂排列。返回值 G 是一个 tf 对象，该对象包含了传递函数的分子和分母信息。

例 13.1　一个传递函数模型

$$G(s)=\frac{s^2+2s+3}{s^4+2s^3+3s^2+4s+5}$$

可以由下面命令输入到 MATLAB 工作空间去：

```
>> num = [1 2 3];den = [1 2 3 4 5];G = tf(num,den)
 Transfer function:
          s^2 + 2 s + 3
 ----------------------------
 s^4 + 2 s^3 + 3 s^2 + 4 s + 5
```

对于传递函数的分母或分子有多项式相乘的情况，MATLAB 提供了求两个向量的卷积函数 ——conv( ) 函数。conv( ) 函数允许任意地多层嵌套，从而表示复杂的计算。应该注意括号要匹配，否则会得出错误的信息与结果。

例 13.2　一个较复杂传递函数模型

$$G(s)=\frac{2(s+2)(s+3)}{(s+1)^2(s+6)(s^3+2s^2+3s+4)}$$

该传递函数模型可以通过下面的语句输入到 MATLAB 工作空间去。

```
>> num=2*conv([1 2],[1 3]);
den=conv(conv(conv([1 1],[1 1]),[1 6]),[1 2 3 4]);
G=tf(num,den)
    Transfer function:
                      2 s^2 + 10 s + 12
-------------------------------------------------------
   s^6 + 10 s^5 + 32 s^4 + 60 s^3 + 83 s^2 + 70 s + 24
```

对于一个 tf 对象,它有自己的属性(域元素),属性值既可以直接获取也可以通过函数 get 来获取。另外可以用函数 set 设置属性值。tf 对象的属性有

```
>> set(tf)
      num: Ny-by-Nu cell of row vectors (Nu = no. of inputs)
      den: Ny-by-Nu cell of row vectors (Ny = no. of outputs)
     Variable: [ 's' | 'p' | 'z' | 'z^-1' | 'q' ]
      Ts: Scalar (sample time in seconds)
      ioDelay: Ny-by-Nu array (I/O delays)
      InputDelay: Nu-by-1 vector
      OutputDelay: Ny-by-1 vector
      InputName: Nu-by-1 cell array of strings
      OutputName: Ny-by-1 cell array of strings
      InputGroup: M-by-2 cell array for M input groups
      OutputGroup: P-by-2 cell array for P output groups
Notes: Array or cell array of strings
UserData: Arbitrary
```

将例 6.2 传递函数算子符号变为 p,延迟时间设为 0.5,可以使用两种 MATLAB 语句来实现

```
G.Variable='P';G.Td=0.5;或
set(G,'Variable','p','Td',0.5);
```

这时再显示 G 时,将得到

```
>> G
 Transfer function:
                                    2 p^2 + 10 p + 12
exp(-0.5*p) * -------------------------------------------------------
                   p^6 + 10 p^5 + 32 p^4 + 60 p^3 + 83 p^2 + 70 p + 24
```

也可用 get()语句来获取属性

```
>> get(G)
num: {[0 0 0 0 2 10 12]}
```

```
den: {[1 10 32 60 83 70 24]}
Variable: 'p'
Ts: 0
ioDelay: 0
InputDelay: 0.5
OutputDelay: 0
InputName: {''}
OutputName: {''}
InputGroup: {0x2 cell}
OutputGroup: {0x2 cell}
Notes: {}
UserData: []
```

13.1.2　零极点模型

零极点模型是描述单变量线性时不变系统传递函数的另一种常用方法，一个给定传递函数的零极点模型一般可以表示为

$$G(s)=k\frac{(s+z_1)(s+z_2)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)}$$

其中 $-z_1,-z_2,\cdots,-z_m$；$-p_1,-p_2,\cdots,-p_n$；k 分别是系统的零点、极点和根轨迹增益。

调用格式：G=zpk (z,p,k)

注意：对单变量系统来说，系统的零极点应该用列向量来表示。

同样，zpk 对象有自己的属性值，该属性值可以用 get()函数来获取，用 set()来设置。具体操作同 tf 对象属性的操作。zpk 对象的属性有

```
>> set(zpk)
   z: Ny-by-Nu cell of vectors (Nu = no. of inputs)
   p: Ny-by-Nu cell of vectors (Ny = no. of outputs)
   k: Ny-by-Nu array of double
   Variable: [ 's' | 'p' | 'z' | 'z^-1' | 'q' ]
  DisplayFormat: [ 'roots' | 'time-constant' | 'frequency' ]
   Ts: Scalar (sample time in seconds)
   ioDelay: Ny-by-Nu array (I/O delays)
   InputDelay: Nu-by-1 vector
   OutputDelay: Ny-by-1 vector
   InputName: Nu-by-1 cell array of strings
   OutputName: Ny-by-1 cell array of strings
   InputGroup: M-by-2 cell array for M input groups
   OutputGroup: P-by-2 cell array for P output groups
   Notes: Array or cell array of strings
   UserData: Arbitrary
```

例 13.3　假设系统的零极点模型为

$$G(s)=2\frac{(s+2)(s+1\pm j1)}{(s+\sqrt{2}\pm j\sqrt{2})(s-3.9765\pm j0.0432)}$$

则该模型可以由下面语句输入到 MATLAB 工作空间去。

```
>> k=2;
z=[-2;-1+j;-1-j];
p=[-1.4142+1.4142*j;-1.4142-1.4142*j;
   3.9765+0.0432*j;3.9765-0.0432*j];
G=zpk(z,p,k)
 Zero/pole/gain:
           2 (s+2) (s^2   + 2s + 2)
---------------------------------------
(s^2   - 7.953s + 15.81) (s^2   + 2.828s + 4)
```

13.1.3　状态方程模型

状态方程式描述系统动态模型的另外一种方法，它不但适合于线性模型，也适于描述非线性模型。

由一个例子引出状态方程模型：

$$\frac{Y(s)}{U(s)}=G(s)=\frac{9}{s^2+3s+9}$$

其微分方程为

$$\ddot{y}+3\dot{y}+9y=9u$$

若令 $x_1=y$，$x_2=\dot{y}$，则有

$$\begin{bmatrix}\dot{x}_1\\\dot{x}_2\end{bmatrix}=\begin{bmatrix}0&1\\-9&-3\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}+\begin{bmatrix}0\\9\end{bmatrix}u$$

$$y=\begin{bmatrix}1&0\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}+0u$$

对于线性时不变系统来说，其状态方程为

$$\begin{cases}\dot{\boldsymbol{x}}=\boldsymbol{Ax}+\boldsymbol{Bu}\\\boldsymbol{y}=\boldsymbol{Cx}+\boldsymbol{Du}\end{cases}$$

在 MATLAB 下只需将各系数矩阵输入到工作空间即可。

调用格式：G = ss(A,B,C,D)

同样可以用 set(ss) 得到状态方程的所有域元素细节，get(G) 得到模型的域值。

例 13.4　双输入双输出系统的状态方程表示为

$$\dot{\boldsymbol{x}}=\begin{bmatrix}1&2&0&4\\3&-1&6&2\\5&3&2&1\\4&0&-2&7\end{bmatrix}\boldsymbol{x}+\begin{bmatrix}2&3\\1&0\\5&2\\1&2\end{bmatrix}\boldsymbol{u}$$

$$\boldsymbol{y}=\begin{bmatrix}0&0&2&1\\2&2&0&1\end{bmatrix}\boldsymbol{x}$$

该状态方程可以由下面语句输入到 MATLAB 工作空间去。

```
>> A=[1,2,0,4;3,-1,6,2;5,3,2,1;4,0,-2,7];
B=[2,3;1,0;5,2;1,1];
C=[0,0,2,1;2,2,0,1];
D=zeros(2,2);
G=ss(A,B,C,D)
a =
        x1   x2   x3   x4
   x1    1    2    0    4
   x2    3   -1    6    2
   x3    5    3    2    1
   x4    4    0   -2    7
 b =
        u1   u2
   x1    2    3
   x2    1    0
   x3    5    2
   x4    1    1
 c =
        x1   x2   x3   x4
   y1    0    0    2    1
   y2    2    2    0    1
 d =
        u1   u2
   y1    0    0
   y2    0    0
 Continuous-time model.
```

13.2 模型的基本结构

在实际应用中，系统的模型通常是由相互连接的模块构成的，本节将介绍相互连接的系统结构的总模型求取方法。

13.2.1 串联连接结构

在串联连接下(见图 13.1(a))，整个系统的传递函数为 $G(s)=G_2(s)G_1(s)$。对单变量系统来说，这两个模块是可以互换的，对多变量系统来说，一般不具备这样的关系。

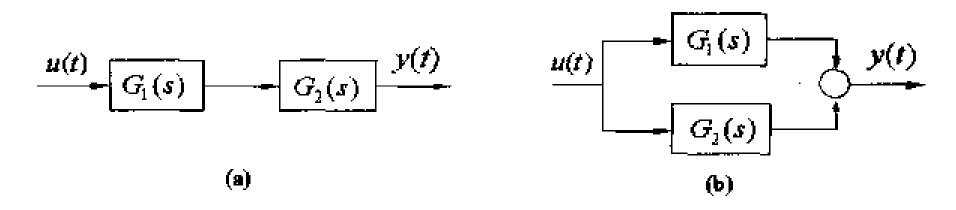


图 13.1　模块的信号连接

(a) 串联结构；(b) 并联结构

假设在 MATLAB 下第一个模块 $G_1(s)$ 的 LTI 对象为 G_1（它可以由 tf,ss 和 zpk 中任意的形式给出），而第二个模块 $G_2(s)$ 的 LTI 对象为 G_2，则整个串联系统的 LTI 模型可以由下列 MATLAB 命令得出

$$G=G_1*G_2;$$

13.2.2　并联连接结构

在并联连接下（见图 13.1(b)），整个系统的传递函数为 $G(s)=G_1(s)+G_2(s)$。

假设在 MATLAB 下第一个模块 $G_1()$ 的 LTI 对象为 G_1（它可以由 tf,ss 和 zpk 中任意的形式给出），而第二个模块 $G_2(s)$ 的 LTI 对象为 G_2，则整个串联系统的 LTI 模型可以由下列 MATLAB 命令得出

$$G=G_1+G_2;$$

13.2.3　反馈连接结构

两个模块 $G_1(s)$ 和 $G_2(s)$ 正、负反馈连接后（见图 13.2），系统总的模型分别为

$$G(s)=\frac{G_1(s)}{1\mp G_1(s)G_2(s)}$$

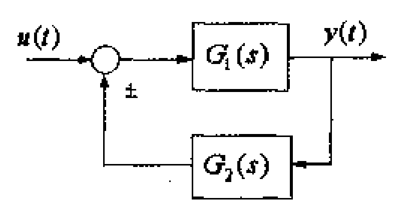


图 13.2　反馈连接结构

控制系统工具箱提供了 feedback() 函数，用来求取反馈连接下总的系统模型。

调用格式：G=feedback（G_1,G_2,sign）

其中变量 sign 为 −1（或 +1）表示负反馈（或正反馈），缺省为负反馈结构。G_1,G_2 分别为前向、反向模型的 LTI 对象，G 为总系统模型。

例 13.5　有两个模型 $G_1(s)=\dfrac{1}{(s+1)^2}$,$G_2(s)=\dfrac{s+2}{(s+3)(s+4)}$，如果采用负反馈结构可以用下面的 MATLAB 语句得到整个系统的传递函数模型。

```
>> G1=tf(1,[1,2,1]);
G2=tf([1,2],[1,7,12]);
G=feedback(G1,G2)
 Transfer function:
              s^2 + 7 s + 12
 ------------------------------------
  s^4 + 9 s^3 + 27 s^2 + 32 s + 14
```

若采用正反馈连接结构，则得出下面结果

```
>>G=feedback(G1,G2,+1)
 Transfer function:
             s^2 + 7 s + 12
 ---------------------------------
 s^4 + 9 s^3 + 27 s^2 + 30 s + 10
```

13.2.4 复杂系统的传递函数求取

控制系统工具箱提供了一个 . m 函数 connect( )和一个 . m 文件 blkbuild 来求取含有相互连接模块的模型。具体的求取过程如下：

(1) 将通路排号；

(2) 用 blkbuild 文件建立原始模型的增广状态方程模型；

(3) 建立连接关系矩阵 $\boldsymbol{Q}$；

(4) 用 connect 建立整个系统的模型。

13.3 不同模型对象的相互转换和模型数据的还原

13.3.1 模型对象的相互转换

LTI 对象模型可以用不同形式描述，它们之间可以相互转换，转换关系如图 13.3 所示。

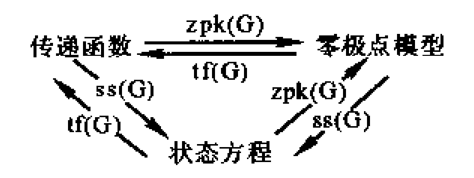


图 13.3 模型对象之间转换关系

13.3.2 模型数据的还原

前面我们学习了建立连续 LTI 系统模型的 tf ,zpk,ss 函数，MATLAB 还提供了相应的函数可以把建立模型时的数据(输入参数)还原出来。这些函数的用法如下：

```
[num,den]=tfdata( G )
[z,p,k]=zpkdata( G )
[A,B,C,D]=ssdata(G)
```

显示还原变量的数据用

```
[num,den]=tfdata( G ,'v')
[z,p,k]=zpkdata( G ,'v')
[A,B,C,D]=ssdata(G,'v')
```

例 13.6　还原例 13.5 负反馈模型数据，可用下面的 MATLAB 语句

```
    >> [num,den]=tfdata( G ,'v')
num =
     0     0     1     7    12
den =
     1     9    27    30    10
    >> [z,p,k]=zpkdata( G ,'v')
z =
    -4
    -3
p =
   -3.6180
   -3.4142
   -1.3820
   -0.5858
k =
     1
```

13.4　控制系统分析与设计

13.4.1　控制系统的线性分析

1. 线性时不变系统浏览器 LTI Viewer 介绍

使用 LTI　Viewer 进行系统的线性分析时，在默认情况下，LTI Viewer 浏览器窗口所显示的图形为系统在单位阶跃信号作用下的系统响应。LTI Viewer 浏览器提供了极其丰富的功能，它可以使用户对系统进行非常详细的线性分析。下面以传递函数 $G(s)=\frac{s+2}{s^2+2s+3}$ 为例对 LTI Viewer 进行详细的介绍与说明。

(1) 绘制系统的不同响应曲线。在默认的情况下，LTI Viewer 绘制系统在单位阶跃信号输入下的系统响应曲线(即阶跃响应)。使用 LTI Viewer 可以绘制不同的系统响应，在 LTI Viewer 图形绘制窗口中单击鼠标右键，选择弹出菜单 Plot Type 下的子菜单，可以在 LTI Viewer 图形绘制窗口中绘制不同的系统响应曲线，如图 13.4 所示。

如果用户选择 Impulse 命令，则可以绘制系统的单位脉冲响应曲线，如图 13.5 所示。

除此之外，使用 LTI Viewer 还可以绘制系统的波特图(Bode)、波特图幅值图(Bode Magbuitude)、奈奎斯特图(Nyquist)、尼科尔斯图(Nichols)、奇异值分析(Singular Value)以及零极点图(Pole/Zero)等，其方法与绘制脉冲响应一致。

(2) 改变系统响应曲线绘制布局。在默认的情况下，LTI Viewer 图形绘制窗口中仅仅绘制一个系统响应曲线。如果用户需要同时绘制多个系统响应曲线图，则可以使用 LTI Viewer

窗口中 Edit 菜单下的 Plot configurations 对 LTI Viewer 图形绘制窗口的布局进行改变，并在指定的位置绘制指定的响应曲线。图 13.6 为响应曲线绘制布局设置对话框，以及采用图中给出的设置同时绘制 6 幅不同的响应曲线。用户可以选择 LTI Viewer 所提供的 6 种不同的绘制布局，在指定的区域绘制自己想要的响应曲线。

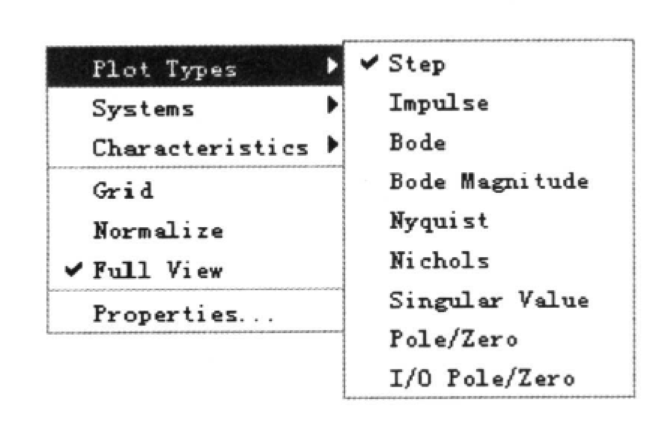


图 13.4 系统响应曲线绘制选择

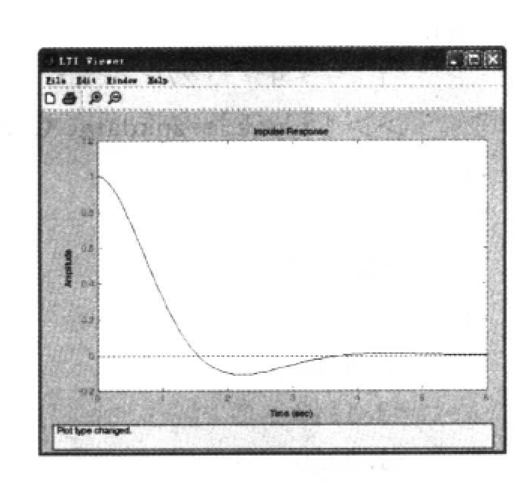


图 13.5 控制系统单位脉冲响应曲线

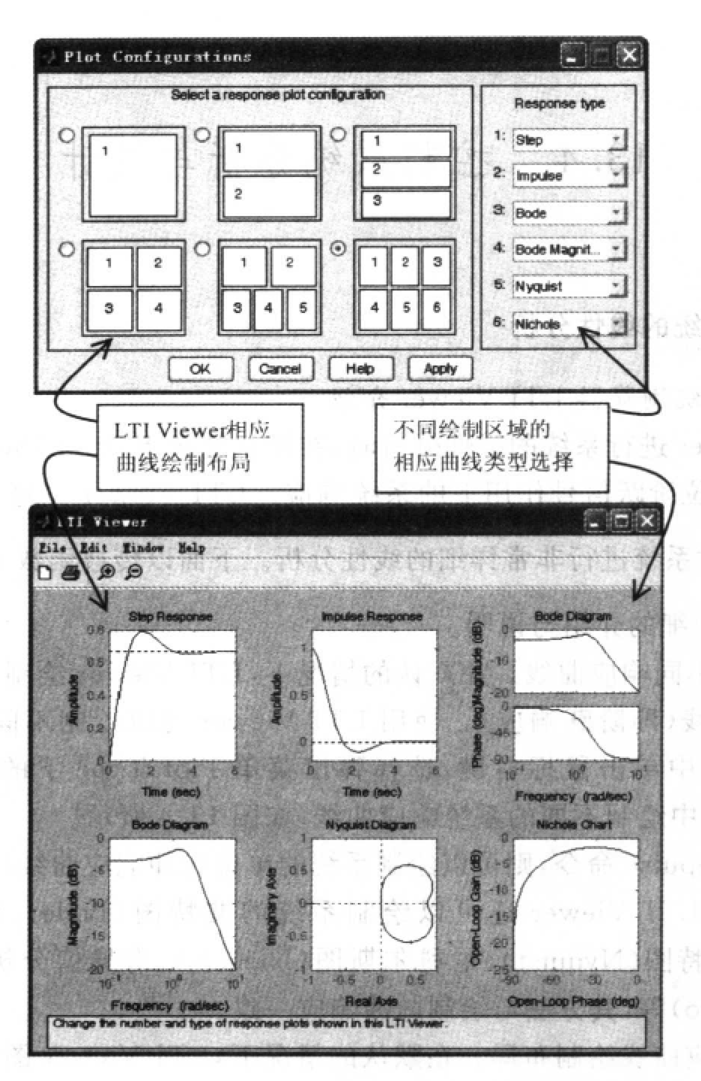


图 13.6 响应曲线布局设计及绘制结果

(3) 系统时域性能分析。使用 LTI Viewer 不仅可以方便地绘制系统的各种响应曲线，还可以从系统响应曲线中获得系统响应信息，从而使用户可以对系统性能进行快速地分析。首先，通过单击系统响应曲线上任意一点，可以获得动态系统在此时刻的所有信息，包括运行系统的名称，以及其他与此响应类型相匹配的系统性能参数。以传递函数 $G(s)=\dfrac{s+2}{s^2+2s+3}$ 的控制系统的单位阶跃响应为例，单击响应曲线中的任意一点，可以获得系统响应曲线上此点所对应的系统运行时刻(Time)、系统输入值(Amplitude)等信息，如图 13.7 所示。

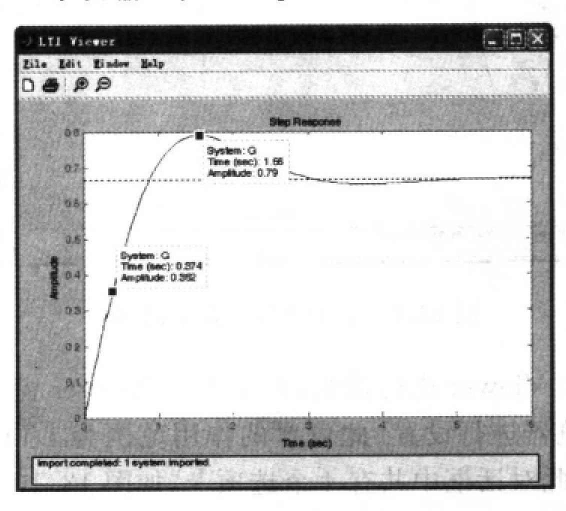


图 13.7 从系统响应曲线获得系统运行信息

其次，用户可以在 LTI Viewer 图形绘制窗口中单击鼠标右键，使用右键弹出菜单中的 Characteristics 子菜单获得系统不同响应的特性参数，对于不同的系统响应类型，Characteristics 菜单的内容并不相同。图 13.8 所示为阶跃响应的特性参数。

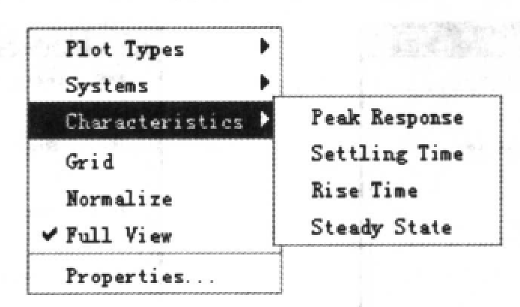


图 13.8 阶跃响应的特性参数

选择 Characteristics 右键弹出菜单中的 Setting Time 可以获得系统阶跃响应的调节时间。此时在 LTI Viewer 绘制的阶跃响应曲线中将出现调节时间标记点，单击此标记点即可获得调节时间，如图 13.9 所示。

对于不同类型的系统响应曲线而言，用来描述响应特性的参数各异。虽然不同响应曲线的特性参数不相同，但是均可以使用类似的方法从系统响应曲线中获得相应的信息。

(4) LTI Viewer 图形界面的高级控制。前面简单介绍了 LTI Viewer 响应曲线绘制窗口的布局设置。Simulink 最为突出的特点之一就是其强大的图形功能。在 Simulink 中，任何图形都是特定的对象，用户可以对其进行强有力的操作与控制。下面介绍如何对 LTI Viewer 图形窗口进行更为高级的控制。

对 LTI Viewer 图形窗口的控制有两种方式。

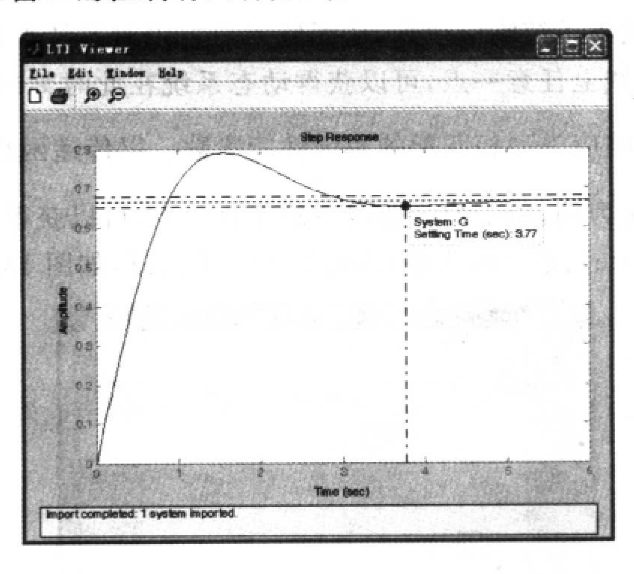


图 13.9　阶跃响应的调节时间

一是对整个浏览器窗口 Viewer 进行控制：单击 LTI Viewer 窗口的 Edit 菜单下的 Toolbox Preferences 命令对浏览器进行设置(此设置的作用范围为 LTI Viewer 窗口以及所有系统响应曲线绘制区域)。在此对话框中共有 4 个选项卡，如图 13.10 所示：

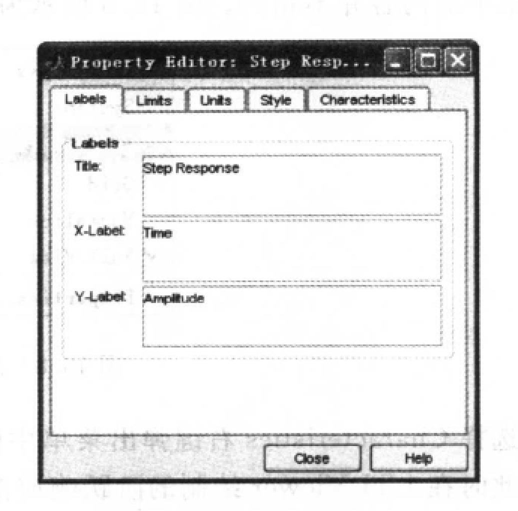
(1) Units 选项：设置图形显示时频率、幅值以及相位的单位。

(2) Style 选项：设置图形显示时的字体、颜色以及绘图网格。

(3) Characteristics 选项：设置系统响应曲线的特性参数。

(4) Parameters 选项：设置系统响应输出的时间变量与频率变量。





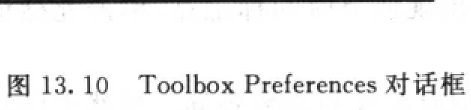


图 13.10　Toolbox Preferences 对话框　　　　图 13.11　Properties 对话框

二是对某一系统响应曲线绘制窗口进行操作：在系统响应曲线绘制窗口中单击鼠标右键，选择弹出菜单中的 Properties 对指定响应曲线的显示进行设置。此对话框中共有 5 个选项卡，如图 13.11 所示：

(1) Labels 选项：设置系统响应曲线图形窗口的坐标轴名称、窗口名称。

(2) Limits 选项：设置坐标轴的输出范围。

(3) Units 选项：设置系统响应曲线图形窗口的显示单位。

(4) Style 选项：设置系统响应曲线图形窗口的字体、颜色以及绘制网格。

(5) Characteristics 选项：设置系统响应曲线的特性参数。

注意：对于不同的系统响应曲线，其特性参数不相同，故 Characteristics 选项卡中内容也不相同。

2. LTI 线性时不变系统对象介绍

LTI 对象有如下的 3 种方式：

(1) ss 对象：封装了由状态空间模型描述的线性时不变系统的所有数据。

(2) tf 对象：封装了由传递函数模型描述的线性时不变系统的所有数据。

(3) zpk 对象：封装了由零极点模型描述的线性时不变系统的所有数据。

(1) LTI 对象的属性。不同的 LTI 对象除了拥有某些共同的属性之外，还有属于每一种对象本身的特殊属性。使用 get 命令，可以获得 LTI 对象的所有属性。仍以 $G(s)=\frac{s+2}{s^2+2s+3}$ 为例。

```
>>get(G)
num: {[0 1 2]}
den: {[1 2 3]}
  Variable: 's'
        Ts: 0
   ioDelay: 0
 InputDelay: 0
OutputDelay: 0
  InputName: {''}
 OutputName: {''}
 InputGroup: [1x1 struct]
OutputGroup: [1x1 struct]
      Notes: {}
   UserData: []
```

其中从 Ts 开始之后的属性为所有 LTI 对象均具有的属性，分别用来描述 LTI 系统的采样时间、输入输出延迟、输入输出端口名称以及其他用户自定义的数据，等等。而在 Ts 之前的属性则属于不同对象本身所特有的，用来描述线性时不变系统。

相应地，使用 set 命令可以对 LTI 对象的指定属性进行修改，其使用方法与设置系统模型或其中的系统模块的属性相类似。

(2) 对 LTI 对象的基本操作。由于 LTI 对象是控制工具箱中最基本的数据类型，因而 MATLAB 支持对 LTI 对象的直接操作。用户可以使用控制工具箱中的系统分析设计命令对这些 LTI 对象进行操作。而且，由于 LTI 对象包括线性系统是连续还是离散的信息，因此，可以使用同样的命令对连续系统与离散系统进行操作。下面介绍 LTI 对象本身的一些简单操作。

1) 生成 LTI 对象。使用 ss，tf 及 tpk 可以建立不同类型的 LTI 对象，如使用 tf 命令建立

使用传递函数描述的线性时不变系统对象。

```
>>mysys_tf=tf([1 2], [1 2 3])      %生成 tf 对象 mysys_tf。
Transferfunction:
      s + 2
-------------
 s^2 + 2 s + 3
```

2) LTI 对象间的相互转换。同样可以使用 ss, tf 及 zpk 进行 LTI 对象之间的相互转换，如

```
>>mysys_ss=ss(mysys_tf)      %将 tf 对象转换为 ss 对象。
a =
            x1      x2
   x1       -2   -0.75
   x2        4       0
b =
        u1
   x1    1
   x2    0
c =
         x1    x2
   y1     1   0.5
d =
        u1
   y1    0
Continuous-time model.      %指明系统为连续时间系统。
```

3) 线性时不变系统的并联，即 LTI 对象的相加，如

```
>> sys1=tf([1 2],[1 2 3]);      %生成系统 1。
>> sys2=tf([1 1],[3 2 -1]);      %生成系统 2。
>>sys=sys1+sys2      %并联系统 1 与 2。
Transfer function:
      4 s^3 + 11 s^2 + 8 s + 1
---------------------------------
 3 s^4 + 8 s^3 + 12 s^2 + 4 s - 3
```

13.4.2 线性控制系统分析设计

在控制系统的分析设计之中，线性系统的设计、仿真分析与实现具有重要的地位。在 MATLAB 中所提供的控制系统工具箱对控制系统的设计提供了强大的支持，用户可以使用控制系统工具箱设计与分析控制系统，然后使用 Simulink 对所设计的控制系统进行仿真分析，并在需要的情况下修改控制系统的设计以达到特定的目的，从而使得用户快速完成系统设计的任务，大大提高设计的效率。

1. 控制系统工具箱简介

控制系统工具箱是MATLAB中所提供的对控制系统进行辅助设计的功能强大的开发设计工具。它包含了丰富的线性系统分析和设计函数，并以LTI对象为基本数据类型对线性时不变系统进行操作与控制。控制系统工具箱能够完成系统的时域和频域分析。在控制系统工具箱中，可以使用不同的方法设计线性反馈系统，如：

(1) 根轨迹设计分析法。

(2) 极点配置法。

(3) H_2和H_∞控制。

(4) 状态观测器设计。

(5) 规范型实现设计。

在使用控制系统工具箱完成线性反馈系统设计之后，便可以通过Simulink进行系统的动态仿真，从而得到真实的、非线性系统的响应，进一步对控制器进行验证。

2. 系统分析与设计简介

控制系统工具箱中最基本的数据类型为LTI对象。无论LTI对象的类型如何，都可以使用相同的命令对其进行分析，因为LTI对象包含了线性时不变系统的所有信息。这里简单介绍一下用来对由LTI对象所描述的线性时不变系统进行分析设计的命令函数。

(1) 动态分析函数。动态分析函数有pole(sys)，dcgain(sys)，tzero(sys)，damp(sys)及norm(sys)等等。对于由如下命令：

```
>> mysys_tf=tf([1 2],[1 2 3]);
```

生成的LTI对象mysys_tf所描述的线性时不变系统，可以使用下述函数对其进行分析：

例如：

```
>>pole(mysys_tf)      %求取系统极点。
ans =
-1.0000 + 1.4142i
-1.0000 - 1.4142i
>>dcgain(mysys_tf)      %求取系统直流增益。
ans =
0.6667
```

(2) 时域与频域分析函数。时域与频域分析函数有step(sys)，bode(sys)，impulse(sys)，nichols(sys)，initial(sys，x0)，nyquist(sys)，lsim(sys，u，t)以及sigma(sys)，等等。例如：

```
>>step(mysys_tf)      %绘制系统的单位阶跃响应曲线。
>>figure,nyquist(mysys_tf)      %在新的图形窗口绘制系统的nyqmst图。
```

使用这两个命令分别绘制线性时不变系统mysys_tf的单位阶跃响应与nyquist图，与LTI Viewer中系统响应曲线的操作相类似，用户可以使用右键弹出式菜单获得系统的时域(或频域)的动态响应(或动态性能)，如图13.12所示。

(3) 补偿器设计。使用控制系统工具箱中的函数还可以进行各种系统的补偿设计，如LQG(Linear-Quadratic-Gaussian线性二次型设计)，Root Locus(线性系统的根轨迹设计)，Pole placement(线性系统的极点配置)以及Observer-based regulator(线性系统观测器设计)等。

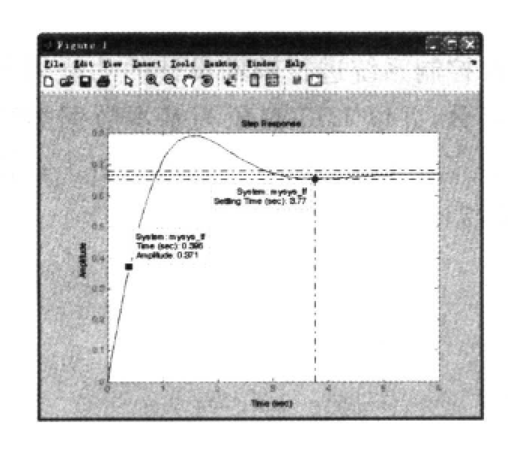
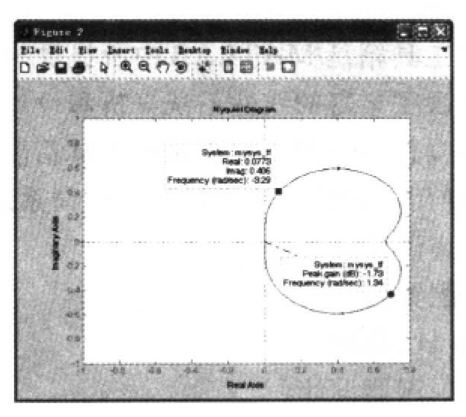



图 13.12 线性时不变系统 mysys_if 的阶跃响应曲线与 nyquist 图

在实际的系统设计中，只要系统经过线性化处理，使用 LTI 线性时不变系统模型来表示，用户都可以使用若干个线性系统控制器的设计方法来进行设计。

3. 单输入单输出系统设计工具

在对非线性系统的线性分析技术进行介绍时，线性时不变系统浏览器 LTI Viewer 是进行系统线性分析的最为直观的图形界面，使用 LTI Viewer 使得用户对系统的线性分析变得简单而直观。其实 LTI Viewer 只是控制系统工具箱中所提供的较为简单的工具，主要用来完成系统的分析与线性化处理，而并非系统设计。

SISO 设计器是控制系统工具箱所提供的一个非常强大的单输入单输出线性系统设计器，它为用户设计单输入单输出线性控制系统提供了非常友好的图形界面。在 SISO 设计器中，用户可以同时使用根轨迹图与波特图，通过修改线性系统零点、极点以及增益等传统设计方法进行 SISO 线性系统设计。下面仍以 tf 对象 mysys_tf 为例说明 SISO 设计器的使用。

(1) 启动 SISO 设计器。在 MATLAB 命令窗口中键入如下的命令启动 SISO 设计器：

```
>>sisotool
```

启动后的 SISO 设计器如图 13.13 所示。

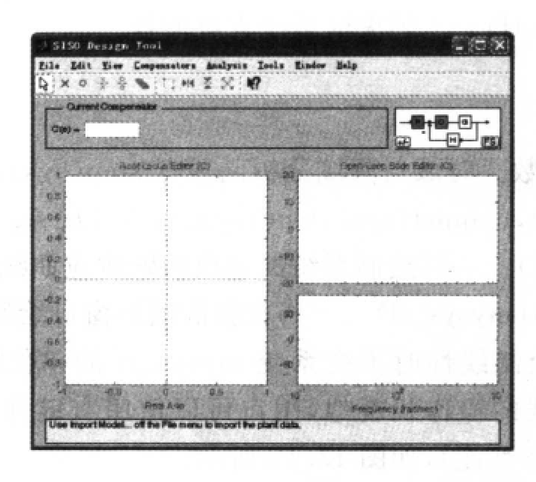


图 13.13 SISO 设计器

在默认的情况下 SISO 设计器同时启用系统根轨迹编辑器与开环波特图编辑器，如图 13.13 所示。

(2) 输入系统数据(Import System Data)。在启动 SISO 设计器之后，需要为所设计的线性系统输入数据，选择 SISO 设计器中 File 菜单下的 Import 命令输入系统数据，此时将打开如图 13.14 所示的对话框。

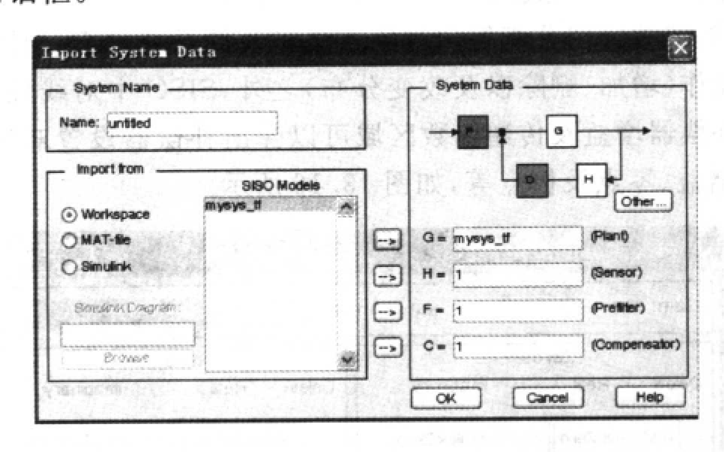


图 13.14　系统数据输入对话框

使用此对话框可以完成线性系统的数据输入。注意，如果数据来源为 Simulink 系统模型框图，则必须对其进行线性化处理以获得系统的 LTI 对象描述。这是因为 SISO 线性系统中的所有对象(G 执行结构、H 传感器、F 预滤波器、C 补偿器)均为 LTI 对象。另外，用户可以单击控制系统结构右下方的 Other 按钮以改变控制系统结构。

使用 SISO 默认的控制系统结构，并设置控制系统的执行结构(即控制对象)数据 G 为 mysys_tf，其他参数如 H，F，C 均使用默认的取值(常数 1)。然后单击 OK 按钮，此时在 SISO 设计器中会自动绘制此负反馈线性系统的根轨迹图以及系统开环波特图，如图 13.15 所示。

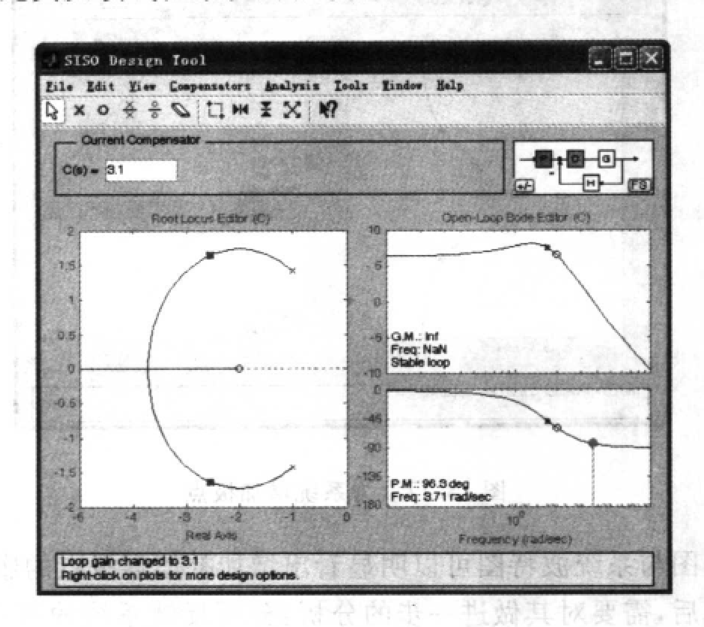


图 13.15　系统数据输入后的 SISO 设计界面

在系统根轨迹图中(见图 13.15)，蓝色“×”和“○”表示控制对象 G 的零极点，而红色表示系统补偿器 C 的零极点。用户可以在根轨迹编辑器中对系统的根轨迹进行控制与操作：增加补偿器的零极点、移动零极点改变其分布、移动根轨迹图中的紫色方块改变系统增益等等，这

些操作均可以改变系统的动态性能。另外，在波特图中除了显示当前补偿器下的系统增益与相位裕度之外，还显示了零点与极点的位置。

(3) 设计与分析系统。在完成线性系统数据的输入之后，用户便可以使用诸如零极点配置、根轨迹分析以及系统波特图分析等传统的方法对线性系统进行设计。除了前面介绍的对系统零极点的各种操作(增加、删除以及改变分布)之外，SISO 中对线性系统的设计提供了诸多的支持，如：单击补偿器增益及传递函数区域可以弹出补偿器设置对话框，使用此对话框可以设置补偿器 C 的增益、零点及极点等，如图 13.16 所示。

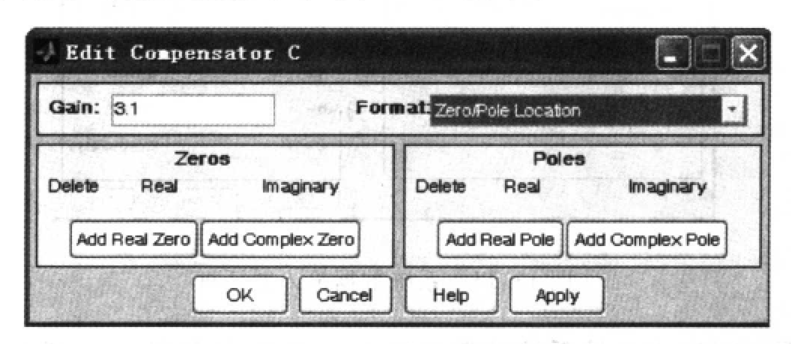


图 13.16 补偿器 C 的增益、零点及极点设置

在此系统设计中，仅仅为补偿器增加一个极点，如图 13.17 所示。

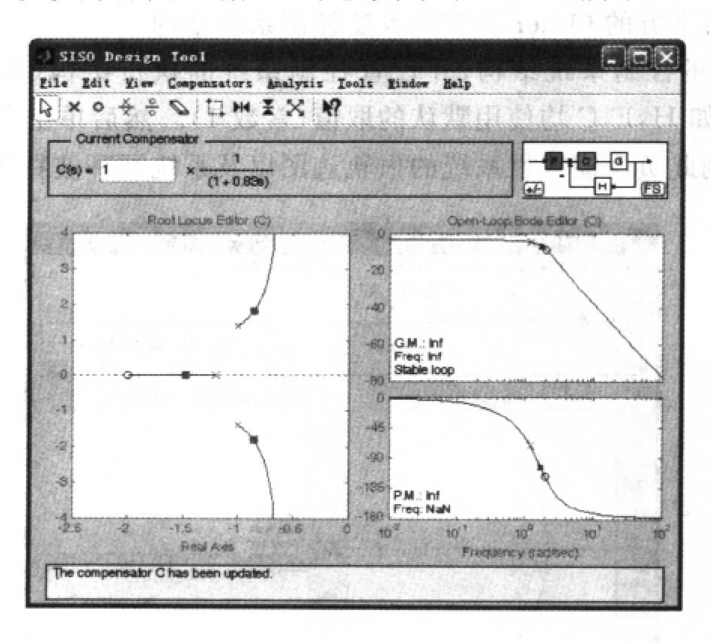


图 13.17 为系统增加极点

从系统的根轨迹图与系统波特图可以明显看出增加补偿器极点的影响。

在系统设计完成后，需要对其做进一步的分析：分析反馈系统的开环和闭环响应，以确保系统是否满足特定的设计要求。用户可以选择 SISO 设计器中 Analysis 菜单下的 Other Loop Responses，绘制指定的开环响应(或闭环响应)曲线。此时将打开 LTI 浏览器，用户可在 LTI 浏览器中对系统的性能如过渡时间、峰值响应、上升时间等等进行分析，如图 13.18 所示。

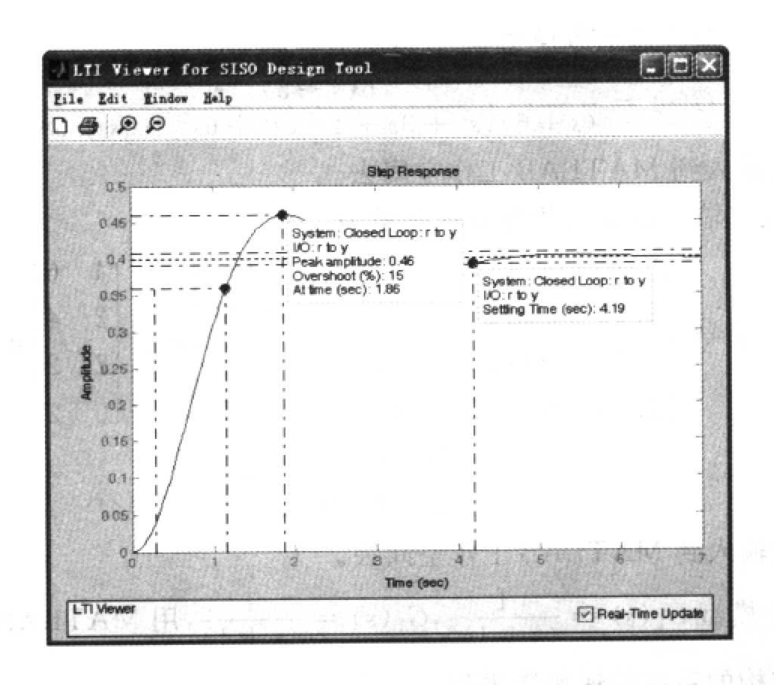


图 13.18　使用 LTI 对系统的设计进行分析验证

如果用户需要设计线性离散控制系统，可以选择 Tools 菜单下的 Continuous/Discrete Conversions 选项，以对离散控制系统的采样时间、连续信号的离散化方法等进行设置，如图 13.19 所示。

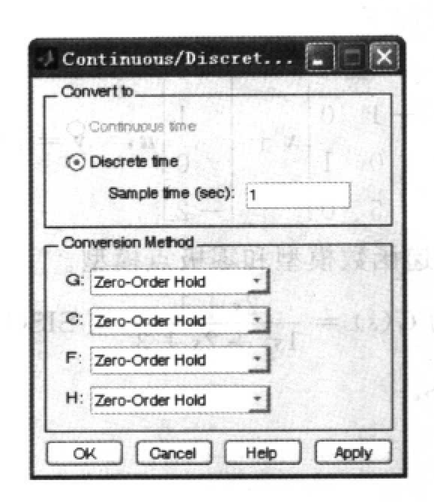


图 13.19　离散控制系统设计：采样时间与离散化方法

生成的 Simulink 系统模型的实现均采用了 MATLAB 工作空间中的 LTI 模块。在生成 Simulink 系统模型之后，便可以对设计好的系统进行仿真分析以验证系统设计的正确性。

习　题

13.1　一个传递函数模型

$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}$$

用 MATLAB 语句输入到工作空间去，并显示零极点模型。

13.2 一个较复杂传递函数模型

$$G(s)=\frac{6(s+5)}{(s+6)(s^2+3s+1)^2(s^3+6s^2+5s+3)}$$

用 MATLAB 语句输入到 MATLAB 工作空间去。

13.3 双输入双输出系统的状态方程表示为

$$\dot{x}=\begin{bmatrix}2.25 & -5 & -1.25 & -0.5\\ 2.25 & -4.25 & -1.25 & -0.25\\ 0.25 & -0.5 & -1.25 & -1\\ 1.25 & -1.75 & -0.25 & -0.75\end{bmatrix}x+\begin{bmatrix}4 & 6\\ 2 & 4\\ 2 & 2\\ 0 & 2\end{bmatrix}u$$

$$y=\begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 2 & 0 & 2\end{bmatrix}x$$

用 MATLAB 语句输入到 MATLAB 工作空间去。

13.4 有两个模型 $G_1(s)=\frac{1}{(s+1)^2}$,$G_2(s)=\frac{1}{(s+1)}$,用 MATLAB 语句得到分别采用负反馈和正反馈结构的系统传递函数模型。

13.5 若典型反馈结构中的 3 个传递函数分别为:

$$G(s)=\frac{s^3+7s^2+24s+24}{s^4+10s^3+25s^2+50s+24},\quad G_c(s)=\frac{10s+5}{s},\quad H(s)=\frac{1}{0.01s+1}$$

用 MATLAB 语句求系统总的传递函数。

13.6 若系统的状态方程如下

$$\dot{x}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & -1 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 5 & 0\end{bmatrix}x+\begin{bmatrix}0\\ 1\\ 0\\ -2\end{bmatrix}u,\quad y=[1\quad 0\quad 0\quad 0]x$$

用 MATLAB 语句求系统的传递函数模型和零极点模型。

13.7 系统的传递函数为 $G(s)=\frac{2s+1}{1s^2+2s+3}$,用 SISO 设计器配置零极点,使超调量不大于 50%,调节时间不大于 4 s。

参考文献

[1] 姚俊，马松辉. Simulink 建模与仿真[M]. 西安：西安电子科技大学出版社，2002.
[2] 薛定宇，陈阳泉. 基于 MATLAB/Simulink 的系统仿真技术与应用[M]. 北京：清华大学出版社，2002.
[3] 薛定宇. 反馈控制系统设计与分析[M]. 北京：清华大学出版社，2000.
[4] 张志涌. 精通 MATLAB(5.3 版)[M]. 北京：北京航空航天大学出版社，2000.
[5] 范影乐，杨胜天，李轶. MATLAB 仿真应用详解[M]. 北京：人民邮电出版社，2001.
[6] 程卫国. MATLAB 5.3 应用指南[M]. 北京：人民邮电出版社，1999.
[7] 张志涌. MATLAB 教程——基于 6.X 版本[M]. 北京：北京航空航天大学出版社，2001.
[8] 徐金明. MATLAB 实用教程[M]. 北京：清华大学出版社，2006.
[9] 郑阿奇，曹弋，赵阳. MATLAB 实用教程[M]. 北京：电子工业出版社，2004.
[10] 沈辉. 精通 SIMULINK 系统仿真与控制[M]. 北京：北京大学出版社，2003.
[11] 魏巍. MATLAB 控制工程工具箱技术手册[M]. 北京：国防工业出版社，2004.
[12] 孙祥，徐流美，吴清. MATLAB 7.0 基础教程[M]. 北京：清华大学出版社，2005.